NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

M5402

VIDEO TRACKING OF OBJECTS ON AN
ENHANCED PC SYSTEM

by

Paul Merritt

December 1989

Thesis Advisor: Roberto Cristi

Approved for public release; distribution is unlimited

T247287

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Naval Postgraduate School; 6b. OFFICE SYMBOL (If applicable): 62	7a. NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code): Monterey, California 93943-5000	7b. ADDRESS (City, State, and ZIP Code): Monterey, California 93943-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
VIDEO TRACKING OF OBJECTS ON AN ENHANCED PC SYSTEM

12. PERSONAL AUTHOR(S)
MERRITT, Paul

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Master's Thesis	FROM _____ TO _____	December 1989	89

16. SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the US Goverment.

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	image processing; edge detection; video tracking; center of mass; autonomous navigation

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This research addressed the problem of fast edge detection for real-time tracking of moving objects. In particular, an implementation suitable to microcomputer application is presented which includes a novel approach of edge detection based on Kalman filtering techniques. Applications to automatic road following show the effectiveness of the technique.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT: ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL: Roberto Cristi	22b TELEPHONE (Include Area Code): 408-646-2223; 22c OFFICE SYMBOL: 62Cx

DD Form 1473, JUN 86 *Previous editions are obsolete* S/N 0102-LF-014-6603

Approved for public release; distribution is unlimited

VIDEO TRACKING OF OBJECTS ON AN ENHANCED PC SYSTEM

by

Paul Merritt
Captain, United States Army
B.S., United States Military Academy, 1982

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL

December 1989

ABSTRACT

This research addresses the problem of fast edge detection for real-time tracking of moving objects. In particular, an implementation suitable to microcomputer application is presented which includes a novel approach of edge detection based on Kalman filtering techniques. Applications to automatic road following show the effectiveness of the technique.

Thesis
M 5402
C.1

TABLE OF CONTENTS

I. INTRODUCTION.....1

II. CENTER OF MASS ALGORITHM FOR OBJECT TRACKING.....4

A. GENERAL.....4

B. SEGMENTATION.....5

C. ALGORITHM.....7

D. LIMITATIONS.....10

1. Lag.....10

2. Concealment.....10

3. Intensity Variation.....10

III. OBJECT TRACKING WITH PREDICTOR.....12

A. GENERAL.....12

B. ALGORITHM..... 12

C. RESULTS.....15

IV. KALMAN FILTER EDGE DETECTION.....18

A. DESIGN APPROACH.....18

B. EDGE DETECTION.....18

C. THEORY.....19

D. DETECTING THE EDGE.....22

E. ANALYSIS OF EDGE DETECTION TRACKING.....25

V. ADJUSTING THE THRESHOLD RANGE.....31

A. GENERAL.....31

B. MOVING AVERAGE.....31

C. LOST OBJECT CALCULATION.....32

D. UNRESOLVED PROBLEMS.....34

VI. AUTONOMOUS NAVIGATION.....36
A. GENERAL.....36
B. THEORY.....36
C. APPLICATION.....38
VII. CONCLUSIONS.....42
APPENDIX A MICROCOMPUTER PACKAGE.....44
A. GENERAL.....44
B. IMAGE PROCESSING SYSTEM.....44
C. MENU DISPLAY AND SELECTION.....45
D. MENU SELECTIONS.....47
1. Read/Save.....47
a. Read.....47
b. Save.....47
2. Display.....47
a. Grab.....47
b. Snap.....47
c. Reinitialize Post-filter.....47
d. Reinitialize Post-threshold.....48
3. LUT.....48
a. Set LUT.....48
b. Store LUT in Array.....48
4. Analysis.....48
a. Histogram.....48
b. Equalize.....48
5. Processing.....48
a. Kalman Filter.....48

b. Track.....49
c. Kalman Edge Detector.....49
d. Autonomous Navigation.....49
e. Threshold.....49
6. Parameters.....49
a. Target Area.....49
b. Covariance - P.....49
c. Covariance - Q.....50
d. Error Tolerance.....50
e. Covariance - R.....50
APPENDIX B MENU.....51
APPENDIX C DATA FILE.....77
APPENDIX D HEADER FILE.....79
REFERENCES.....80
DISTRIBUTION LIST.....81

I. INTRODUCTION

In image processing applications, motion arises from a relative displacement between the video camera and the image being observed. The displacement can result from two situations, either the video camera or viewing platform is in motion (which gives the appearance that all objects or features within the image are moving), or the camera is stationary and the displacement is caused by the movement of an object. Although the video tracking algorithms developed concentrate primarily on the latter situation, application to the case of a moving platform is also investigated.

Tracking of objects and image features from video signals has applications in various areas of robotics, visual guidance systems, anti-aircraft weapons firing systems, and autonomous navigation, to name a few. Most of the algorithms available in the literature are designed to track a desired object based on its edges. Although the edge detection operation is well known, the implementation of using local operators (Sobel, Roberts, Laplacian) [Ref. 1] is computationally inefficient, since it uses the intensity level of each pixel several times. Therefore, applications based on off-the-shelf microcomputers require more efficient algorithms for real-time implementation.

In this thesis we developed a tracking algorithm operating on video signals in real-time, capable of following

objects moving on smooth trajectories. The edge detection technique which separates the objects from the background is based on a variation of the basic Kalman filter which detects changes in the signal model. The efficient use of lookup tables for the Kalman filter gains makes this algorithm attractive for real-time implementation.

The image processing system used to implement the developed tracking algorithms is based on an IBM AT Personal Computer.[1] A PCVISIONplus FRAMEGRABBER[2] board is installed in the computer which allows the use of the ITEX[3] PCplus library of image processing subroutines to develop individual image processing functions. The algorithms and the other image processing functions are implemented in the mouse-driven menu package using the GFX[4] library of graphics subroutines. The PC system configured with these components is relatively inexpensive, portable, and easily modifiable to incorporate a number of image processing functions.

This thesis is organized as follows. An object detection algorithm based on the computation of its center of mass is presented in Chapter II, a Kalman filter predictor algorithm

---

[1] Trademark of IBM

[2] Trademark of Imaging Technology Inc.

[3] Trademark of Imaging Technology Inc.

[4] Trademark of C Source Inc.

used for increased accuracy in the detection is described in Chapter III, and the Kalman filter edge detection technique is discussed in Chapter IV. Chapter V examines several methods of improving the tracking accuracy, Chapter VI investigates the potential of the Kalman filter edge detection technique as an approach to autonomous navigation, and conclusions and recommendations are presented in Chapter VII.

II. CENTER OF MASS ALGORITHM FOR OBJECT TRACKING

A. GENERAL

In this chapter we introduce an algorithm for tracking an object based on the computation of its center of mass. The center-of-mass technique is at the base of most of the developed tracking algorithms. Because of its simplicity, the algorithm can be implemented in real-time at a continuous output rate of nearly thirty frames per second. Even though the number of frames per second is slightly less when the algorithm is included in the processing, its effect is undetectable by the human eye.

Another advantage of the algorithm is that all calculations are done in the spatial domain, rather than the frequency domain. This further reduces processing time and alleviates all memory storage requirements since the images are stored in the framegrabber board.

The real-time implementation of the center of mass computation is based on two operations of the image processor: "snap" and "threshold". In particular,

- Snap - acquires one frame from the video input, stores it in frame memory, and displays it on the monitor.
- Threshold - separates the 256 possible gray levels into two regions, background and object. In our application gray levels within a specified range, the threshold range, are designated as objects while all others are considered background.

B. SEGMENTATION

The operator must have some means of designating which object is to be tracked. Using a function developed by the author, the operator can use the mouse to designate the object. A cursor appears on the video monitor when the algorithm is implemented. The object is selected by using the mouse to position the cursor over it and pressing the left mouse button.

The center of mass algorithm is based on the assumption that the image is binary, in the sense that only one object and background are present. Further, the intensity levels of the object and background must be separable or segmented by a thresholding operation. (The actual number of gray levels present in an image is 256.) The pixel values must be thresholded to reduce the 256 values to two values that can be implemented in the algorithm. A mean intensity value is computed by averaging pixel intensities in the vicinity of the cursor designation. A threshold range is established by adding and subtracting a tolerance level to the mean value. The tolerance level is dependent on the measurement noise, the desired sensitivity, and the extent of the object's homogeneity. The threshold output will have two distinct values, normally zero and one, although these values are application dependent.

Figure 1 depicts the threshold range centered around a calculated mean value. Although the thresholding accounts

for the 256 possible gray levels, it assumes that the object is nearly homogeneous with only slight variations in its intensity. This is a realistic assumption for many objects and is a common assumption even in complex tracking algorithms [Ref. 2].

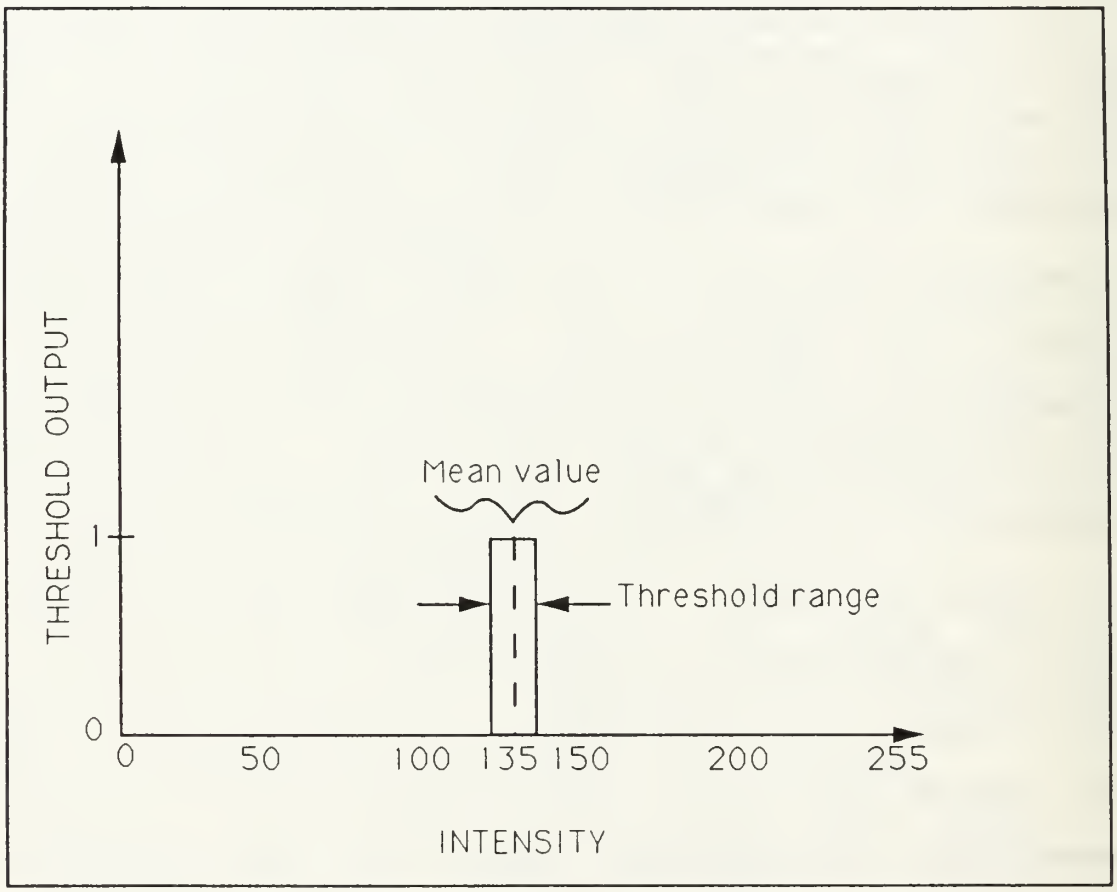

Figure 1. Threshold Output

C. ALGORITHM

In order to perform the computations in real-time, a window is placed on the object to be tracked and the position of the window is updated according to the movement of the object. The window can be centered on the object as shown in Figure 2 since the object's initial location has been designated by the operator. This drastically reduces the number of calculations in the algorithm since only those pixel values within the window need to be examined. As the object moves, a new frame is snapped resulting in an image similar to Figure 3.

The intensity value of each pixel, p(x,y), within the window of the new frame is sequentially examined in order to compute the center of mass of the object. The center of mass of the object is the point that represents the average x coordinate and the average y coordinate for the object. If the pixel belongs to the object, its x and y coordinate values are added to previous values; otherwise the pixel is ignored and the next pixel is examined. After all pixels within the window have been examined, the average x and y coordinates are calculated. This process is shown in the following algorithm.

```
a=b=n=0          (initialize counters)
for x,y = 1,N    (examine only pixels within the window)
if p(x,y) = 0    (pixel(x,y) belongs to the object)
    a = a + x    (add x to coordinates)
```

```
        b = b + y   (add y coordinates)
        n = n + 1   (number of pixels in the object)
      endif
    endfor
```

cm_x = a/n (average x coordinate)

cm_y = b/n (average y coordinate)

The result of the algorithm (cm_x,cm_y) represents the xy coordinates of the center of mass of the object. Using these coordinates, the window can easily be centered on the object's current location as shown in Figure 4. A new frame is snapped and the process is repeated.

Figure 2. Centered Object

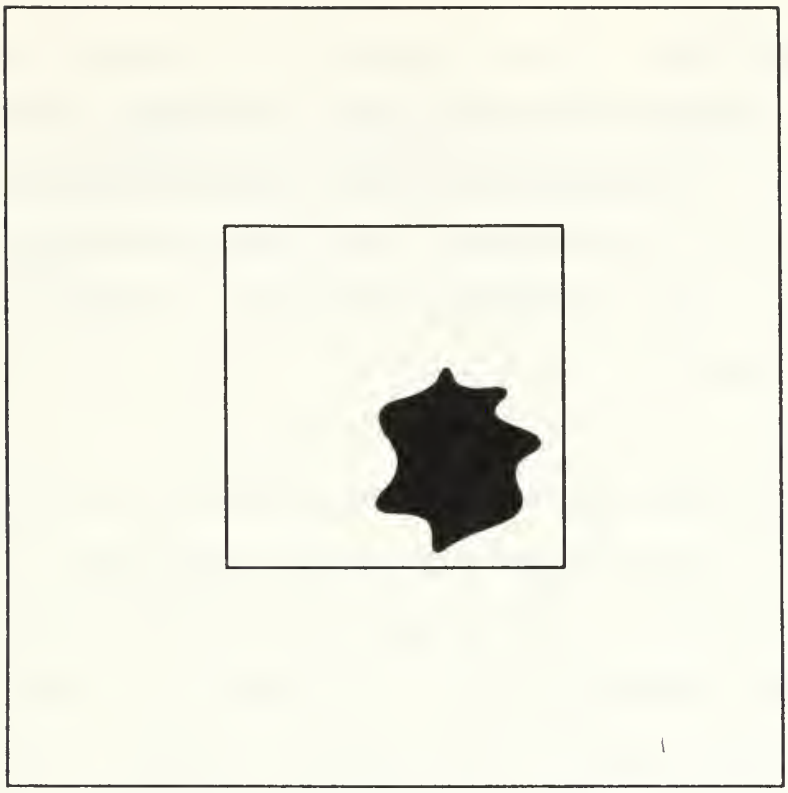

Figure 3. Object After Motion

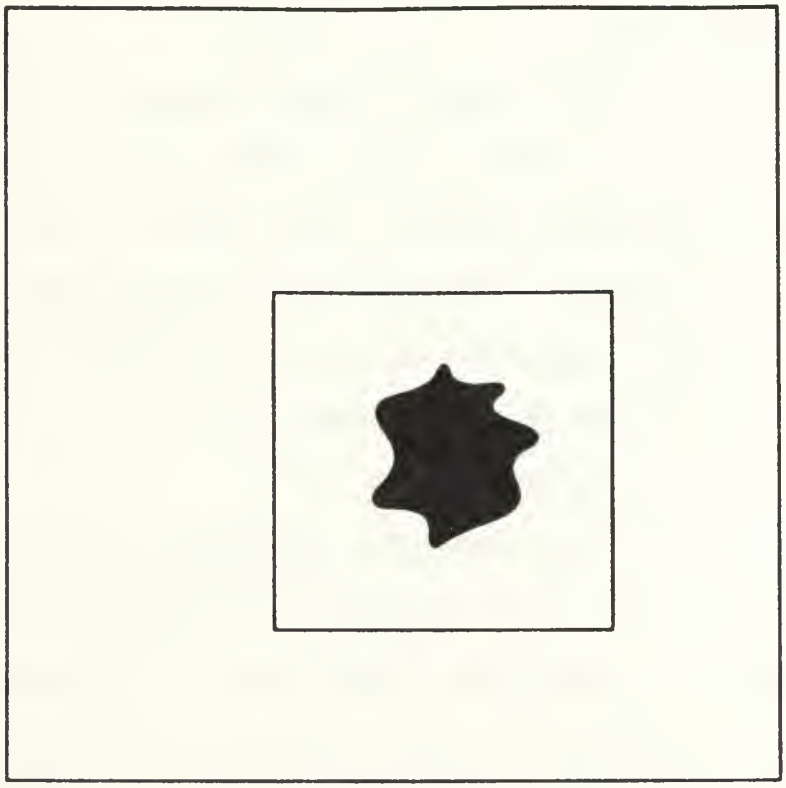

Figure 4. After Center of Mass Calculation

D. LIMITATIONS

The implementation of the algorithms revealed three limitations that required further investigation in order for the tracking to be accurate. The three limitations are: lag, concealment, and intensity variation. Each of these are discussed below.

1. Lag

Although the object is within the window at the completion of each repetition of the algorithm, in general it is not centered. The continuous movement of the object during the computation of the center of mass causes the window to lag behind the object's motion. The predictor introduced in Chapter III solves this problem.

2. Concealment

As the object being tracked passes behind another object, it becomes partially or momentarily obscured. By design, the algorithm centers the window on the unobscured portion of the object. Eventually the centering is no longer accurate and the track is lost. The predictor discussed in Chapter III also solves this problem.

3. Intensity Variation

Changes in the object's intensity or gray levels (as when an object moves from shadows to bright sunlight or vice versa) results in the object being lost. The threshold range is no longer accurate for the object's new gray level. An object moving into a shaded area will change its intensity to

a gray level outside the threshold range. This results in the object being perceived as background. Chapters IV and V investigate possible solutions to this problem.

III. OBJECT TRACKING WITH PREDICTOR

A. GENERAL

In the previous chapter tracking is accomplished using only the data contained in the current frame, without any attempt to use the estimated dynamic motion of the object. This causes the tracking algorithm to lag slightly behind the object's movement and its performance to degrade when the object is temporarily obscured by other objects. These two problems can be eliminated if the future position of the object is estimated accurately. Various prediction algorithms can estimate the position of the object at time t_j using past data from the interval $[t_o , t_k]$ with $t_j > t_k$. Here a Kalman filter predictor is used.

B. ALGORITHM

The future position of the object can be computed by adding the distance traveled to the current position. If the object moves uniformly, the distance traveled by the object during the time interval T can be approximated by the product of velocity and the time interval T. The discrete system that describes the object's motion can be characterized by the following discrete-time equations:

$$Y(k+1) = Y(k) + T * V(k) \quad (1)$$

$$V(k+1) = V(k) + F(k) \quad (2)$$

$$Z(k) = Y(k) + W(k) \quad (3)$$

where

Y(k) is the object's position at the time k (i.e., in the k-th frame).

V(k) is the object's velocity at time k.

Z(k) is the measured position of the object at time k.

F(k) is a random forcing function at time k.

W(k) is a random measurement noise at time k.

T is the time between measurements.

In order to reduce the processing time, the algorithm has been simplified by assuming that the velocity is nearly constant. This is fairly common practice for even complex tracking algorithms [Ref. 2]. The random forcing function F(k) is assumed to have zero mean and a variance of σ_f^2. It represents slight variations in the object's velocity. The measurement noise is assumed to have zero mean with a variance of σ_w^2. It should also be noted that Y(k) and V(k) are actually two-dimensional vectors since each has a horizontal and vertical component. The position and velocity will be left as single states for simplification since it does not affect the results. We can write Equations 1 - 3 in matrix form, as

$$X(k+1) = \phi X(k) + \Gamma F(k) \tag{4}$$

$$Z(k) = CX(k) + W(k) \tag{5}$$

where X(k) is the two dimension state vector for the position and velocity and

$$\phi = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix}$$

$$\Gamma = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

From Equations 4 and 5, we can design a Kalman filter to predict the future position [Ref. 3]. The predictor equations

$$\hat{X}(k|k) = \hat{X}(k|k-1) + G(k)\left[Z(k) - C\hat{X}(k|k-1)\right] \quad (6)$$

$$\hat{X}(k+1|k) = \phi X(k|k) \quad (7)$$

are calculated on-line to predict the new position of the center of mass of the object. The terms in Equation 6 are

$\hat{X}(k|k)$, the estimate of X(k) given measurements at times up to and including k;

$\hat{X}(k|k-1)$, the estimate of x(k) given measurements at times up to and including k - 1; and

G(k), the Kalman filter gain matrix.

Since the Kalman filter gain matrix is independent of the data and depends on the matrices ϕ, Γ, and C only, it can be computed off-line and stored in a lookup table prior to any real-time computation. This gain can also be computed recursively according to

$$G(k) = P(k|k-1)C'\left[CP(k|k-1)C'+R\right]^{-1} \quad (8)$$

$$P(k+1|k) = \phi\left[I - G(k)C\right]P(k|k-1)\phi'+Q \quad (9)$$

where

P(k + 1|k) is the covariance matrix of the error estimate of the state, X(k + 1), given the measurements up to k.

R is the covariance of the measurement noise W(k).

Q is the covariance matrix of the random forcing function F(k).

The matrix P(k|k - 1) is initialized using a priori knowledge of the initial intensity level X(0) and its covariance matrix P(0). In the case of maximum uncertainty, P(0) is assumed to have a large value. The computation is terminated when G(k) reaches steady state. In all cases of our simulations, steady state is normally reached within twenty iterations.

In the real-time application, the Kalman filter gain matrix, G(k), is reinitialized (k = 0) upon reaching the steady state gain in the algorithm. This is done to increase the sensitivity of the system to possible velocity changes in the event that the assumed constant velocity is not always observed.

C. RESULTS

The implementation of the predictor in the tracking algorithm requires only four additional lines of code with the use of lookup tables. The four lines of code represent the predicted horizontal and vertical components of the position and velocity.

Although the introduction of the predictor algorithm increases the processing time, the on-line computations are so few that the tracking algorithm still operates in real-time on the PC. A video signal with various background clutter (e.g., trees) was used to test the reliability of the predictor. The object was free to pass on either side of the clutter. This required the tracker to detect the object after momentary concealment. The predictor algorithm continued to move the window according to the object's last calculated velocity while the object was obscured. As the object emerged from behind the obscuration, it appeared within the window and tracking resumed.

As a result of this algorithm, the window no longer lags behind the object's motion. The tracker is able to detect the object as it emerges from behind the background clutter if the size of the obscuring barrier is limited to approximately one half the size of the window.

Figure 5 shows a plot of the object location in xy coordinates using both the center of mass algorithm and the predictor algorithm. Since the object is moving in the negative direction for both coordinate values, the center of mass plot lies above or lags behind the smooth predictor plot.

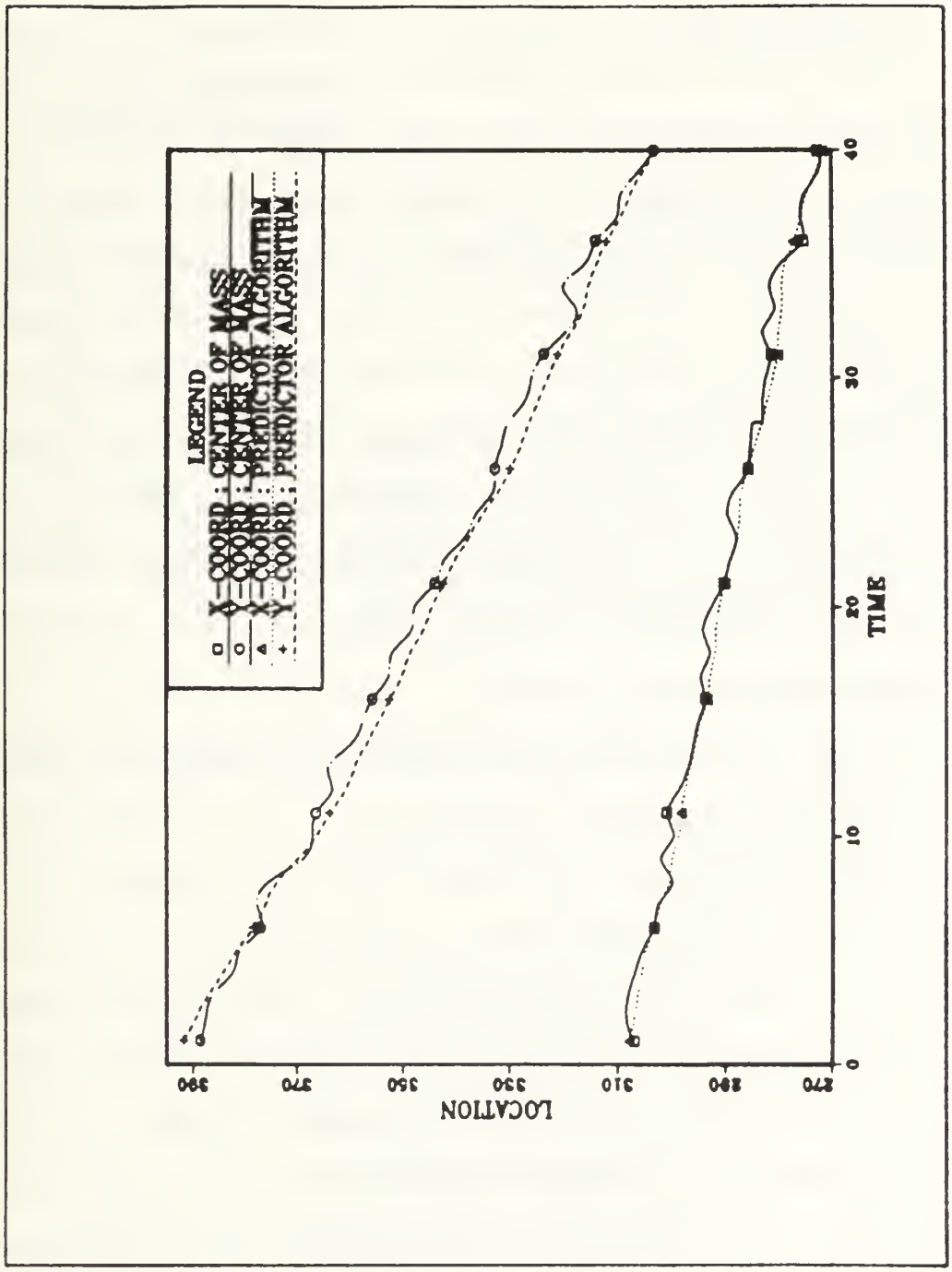

Figure 5. Object Location (Center of Mass vs. Predictor)

IV. KALMAN FILTER EDGE DETECTION

A. DESIGN APPROACH

The previous chapter showed how an object can be tracked by combining segmentation with the computation of the center of mass. Although the results are satisfactory under constant uniform lighting conditions, the presence of shade and variations in the object's luminosity cause the algorithm to lose track of the object. An alternative approach based on edge detection does not degrade in the presence of changes in luminosity. In this chapter, we show a novel implementation of edge detection based on the Kalman filter which can be implemented in real-time.

B. EDGE DETECTION

An edge is defined as the boundary between two regions with distinct gray level intensities. If the edges of the object being tracked are known, they can be used in the center of mass algorithm just as effectively as the entire object but with less computation. Most edge detectors display the edges as bright pixels while non-edge pixels, which are treated as background, appear dark. The edge detector essentially thresholds the image into two distinct gray levels. The center of mass algorithm can then simply be applied to the edge image.

C. THEORY

Most edge detection techniques involve the computation of a local derivative operator. For example, the Laplacian operator is a second-order derivative operator defined as

$$L[f(x,y)] = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{10}$$

where f(x,y) represents an image. The digital Laplacian operator at a pixel, p(x,y), is defined as

$$L[f(x,y)] = x_2 + x_4 + x_6 + x_8 - 4x_5 \tag{11}$$

where x_1 through x_9 represent the gray levels of the pixels in a 3 by 3 neighborhood about p(x,y) shown in Figure 6. The Laplacian edge detection operation is implemented by convolving the 3 x 3 mask of Figure 7 with the image f(x,y). [Ref. 1] This computation is quite time consuming and not well suited for real-time processing.

X_1	X_2	X_3
X_4	X_5	X_6
X_7	X_8	X_9

Figure 6. 3 x 3 Image Region

0	1	0
1	-4	1
0	1	0

Figure 7. Laplacian Mask

A much simpler approach using a Kalman filter is examined. The typical plot of intensity levels on a horizontal line is shown in Figure 8. Basically, the signal is piecewise constant with a small variation or disturbance superimposed; the disturbance accounts for both observation noise as well as details of the object and background. Each

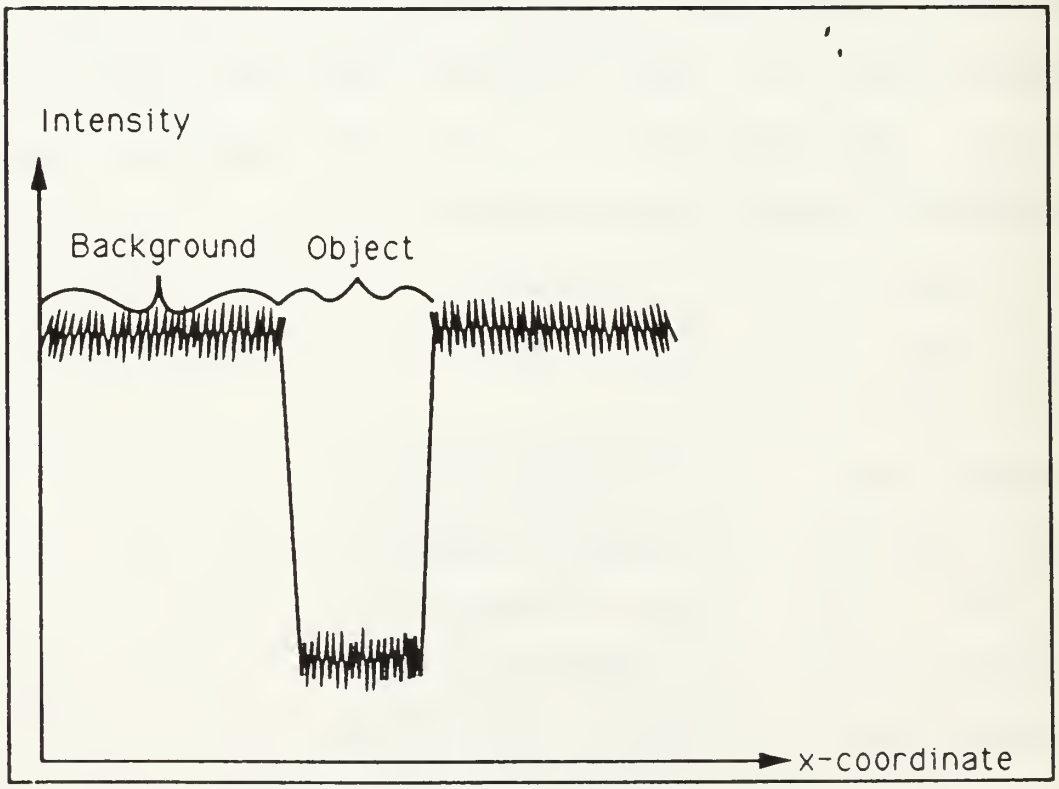

Figure 8. Horizontal Line of Pixels

horizontal line can be represented as a noisy measurement signal, Z(k), where the signal is composed of two distinct regions, background and object. Within each region the true signal, X(k), is nearly constant (of uniform intensity). However, since complete uniformity in an image is unlikely, X(k) will typically have slight variations around the average intensity level. The signal and its measurement for each region can be characterized by the following state and measurement equations:

$$X(k+1) = X(k) + F(k) \tag{12}$$

$$Z(k) = X(k) + W(k) \tag{13}$$

where

X(k) is the true intensity signal.

Z(k) is the noisy measurement.

F(k) is a random forcing function with zero mean and variance σ_f^2 to account for the small variations in intensity.

W(k) is the measurement noise with zero mean and variance σ_w^2.

A Kalman filter can be used to estimate the true signal in each region. Applying the Kalman filter

$$\hat{X}(k|k) = \hat{X}(k|k-1) + G(k)\left[Z(k) - C\hat{X}(k|k-1)\right] \tag{14}$$

$$\hat{X}(k+1|k) = \phi X(k|k) \tag{15}$$

to the signal shown in Figure 8 would result in an estimate similar to Figure 9, a smoothed version of the input signal. The goal of the filtering is to produce an output similar to Figure 10 where the edges are detected and the signal is filtered separately within each region. Once an edge is detected, the filter gains can be reinitialized to begin filtering of the next region. It is necessary, however, to detect the edges while filtering so that the Kalman filter can be reinitialized.

D. DETECTING THE EDGE

A simple algorithm can be incorporated into the filtering to detect the object's edges. The algorithm takes advantage of the Kalman filter's statistical properties discussed below.

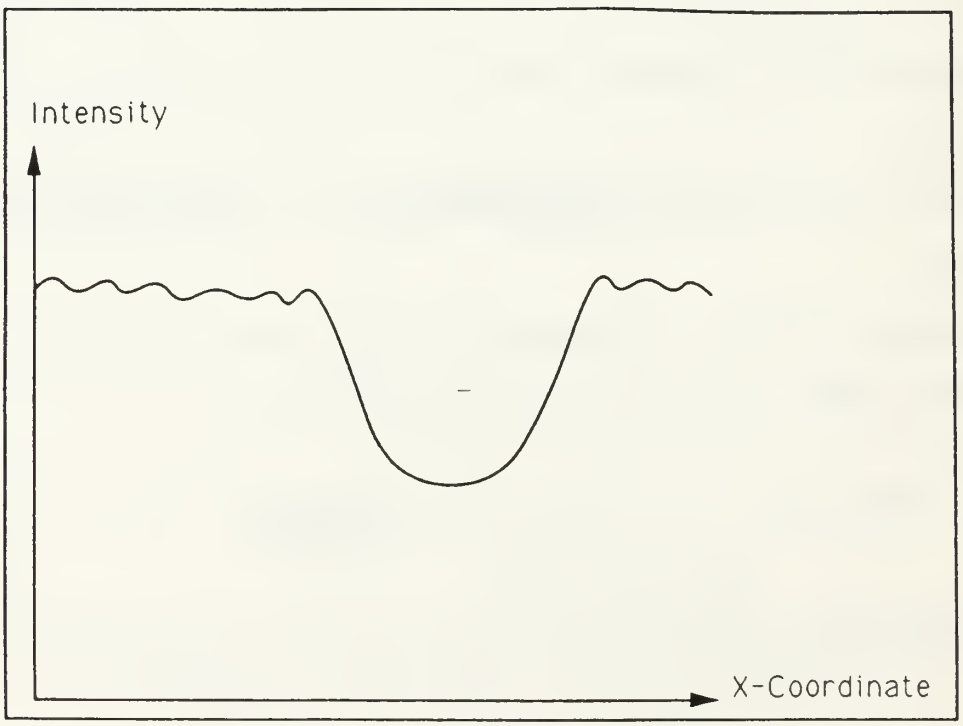

Figure 9. Filtered Signal

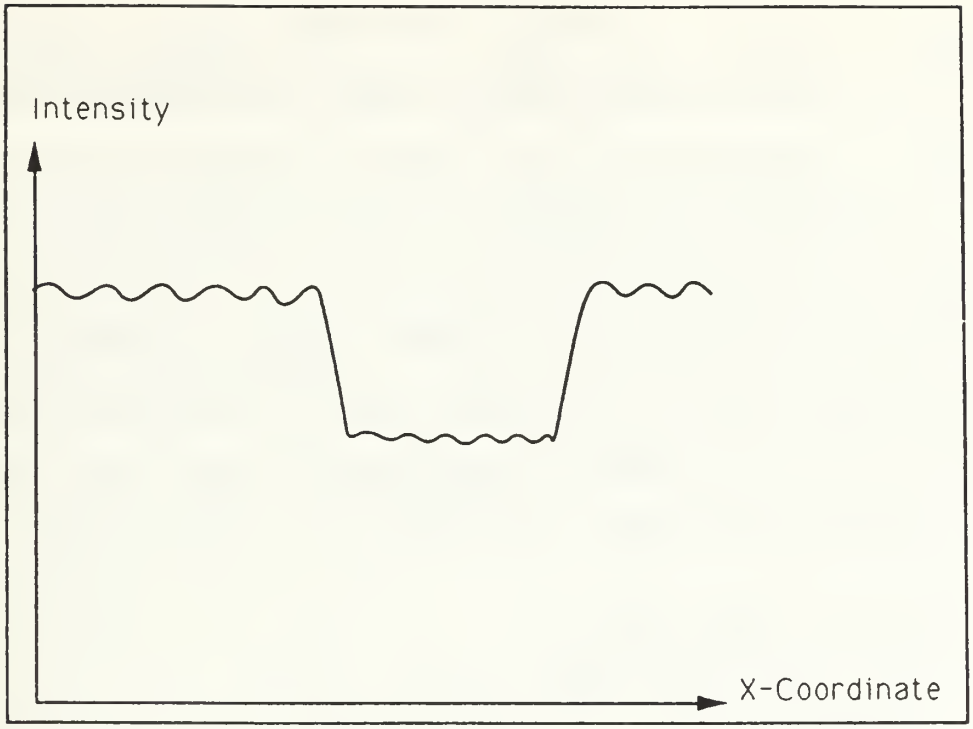

Figure 10. Desired Filtered Signal

Given the state space model

$$X(k+1) = \phi X(k) + \Gamma F(k) \tag{16}$$

$$Z(k) = CX(k) + W(k) \tag{17}$$

where F(k) and W(k) are Gaussian, it is possible to show the following results [Ref. 4]:

$$P\left(X(k)\middle|Z(k-1),\ldots,Z(0)\right) = N\left(\hat{X}(k), P(k|k-1)\right) \tag{18}$$

$$P\left(Z(k)\middle|Z(k-1),\ldots,Z(0)\right) = N\left(C'\hat{X}(k), C'P(k|k-1)C + \sigma_w^2\right) \tag{19}$$

Therefore, given a set of observations Z(0).....Z(k), we can determine the probability that the observations belong to the model by using the conditional density

$$P\left(Z(k)\middle|Z(k-1),\ldots,Z(0)\right)=\frac{1}{\sqrt{2\pi}\left(C'P(k|k-1)C+\sigma_w^2\right)}\exp\left(-\frac{1}{2}\frac{\left|Z(k)-C'\hat{X}(k)\right|^2}{\left(C'P(k|k-1)C+\sigma_w^2\right)}\right)$$

(20)

with X(k) and P(k|k - 1) derived from the Kalman filter. Subtracting the mean and dividing by the standard deviation changes Z(k) to a Gaussian random variable with zero mean and unity standard deviation. We define this value as E(k).

$$E(k)=\frac{Z(k)-C'\hat{X}(k)}{\sqrt{C'P(k|k-1)C+\sigma_w^2}} \tag{21}$$

$$P\left(E(k)\right)=N(0,1) \tag{22}$$

Using statistical theory, confidence intervals for E(k) can now be established. The absolute value of E(k) can be compared to a tolerance level, ϵ, which represents the desired confidence interval. For example, an ϵ value of 2.0 represents a ninety-five percent confidence interval. Therefore, if the absolute value of E(k) is greater than 2.0, then the probability that Z(k) is an edge is greater than 0.95. The algorithm can be stated in the following format:

- Compute E(k).
- If $|E(k)| > \epsilon$, then an edge is detected, reinitialize the Kalman filter.
  Else; no edge detected, update the Kalman filter.
- Continue filtering.

E. ANALYSIS OF EDGE DETECTION TRACKING

The Kalman filter edge detector was analyzed from three perspectives: accuracy, speed, and reliability. Prior to comparing the Kalman filter to other edge detectors, it has been necessary to determine the Kalman filter's ability to accurately detect edges. The algorithm was found to be extremely accurate when the variances of the random forcing function, F(k), and the measurement noise, W(k), are properly chosen; basically by a priori knowledge and trial and error. The algorithm was not found to be overly sensitive to variations of these parameters and it performed satisfactorily for several values of the parameters. One limitation of the Kalman filter as an edge detector is its ability to detect a perfectly horizontal line. Since the image is filtered horizontally, a perfectly horizontal edge will only be detected at each end of the line. This has not proven to be a significant limitation since few objects in the actual tracking had perfectly horizontal edges. Figures 11 and 12 show a typical object before and after edge detection using the Kalman filter.

The speed of the Kalman filter edge detector is far better than anticipated. The tracking algorithm with the Kalman filter edge detector can still be performed in real-time in spite of the increase in processing time due to the Kalman filter edge detector and the center-of-mass algorithm.

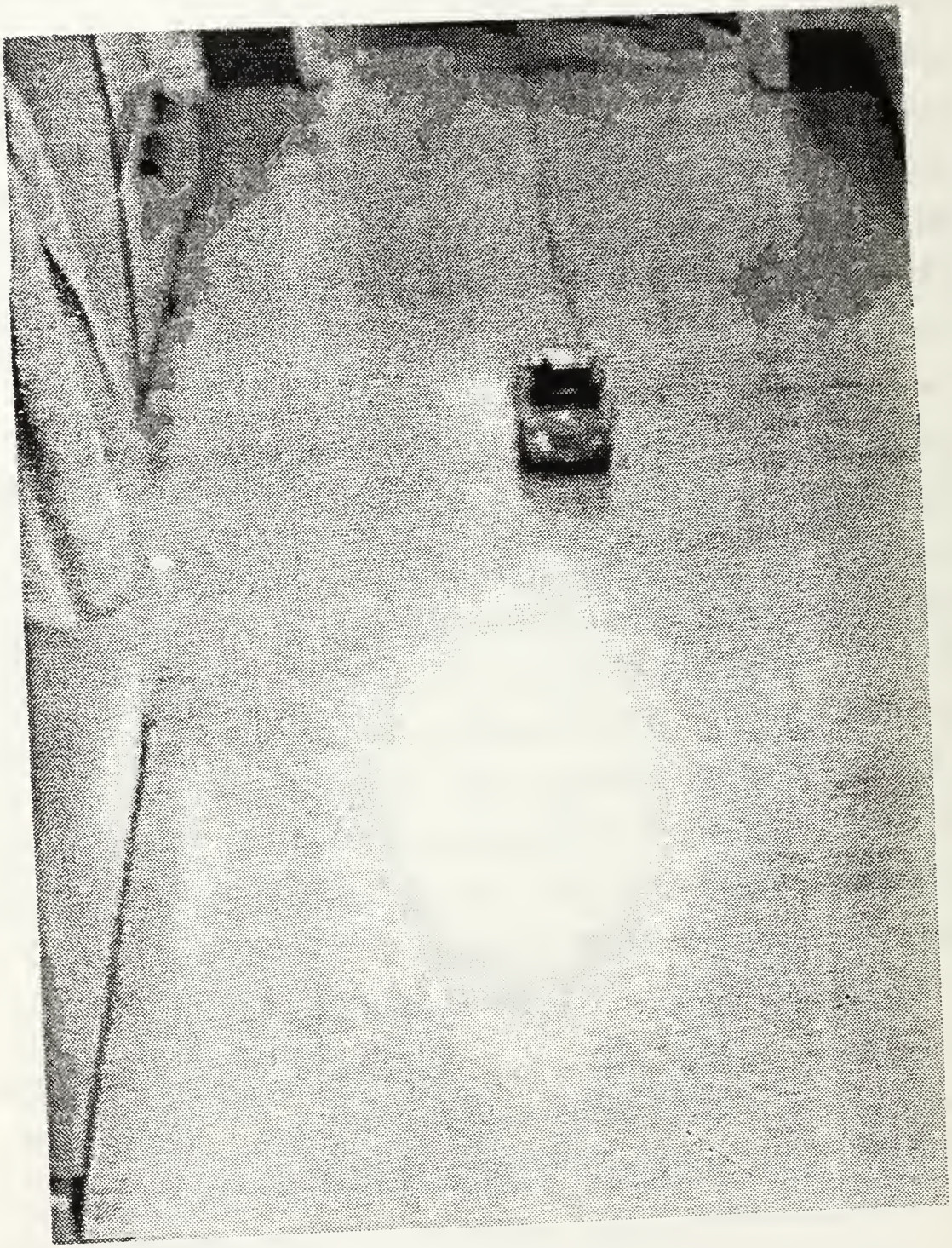

Figure 11. Object Before Edge Detection

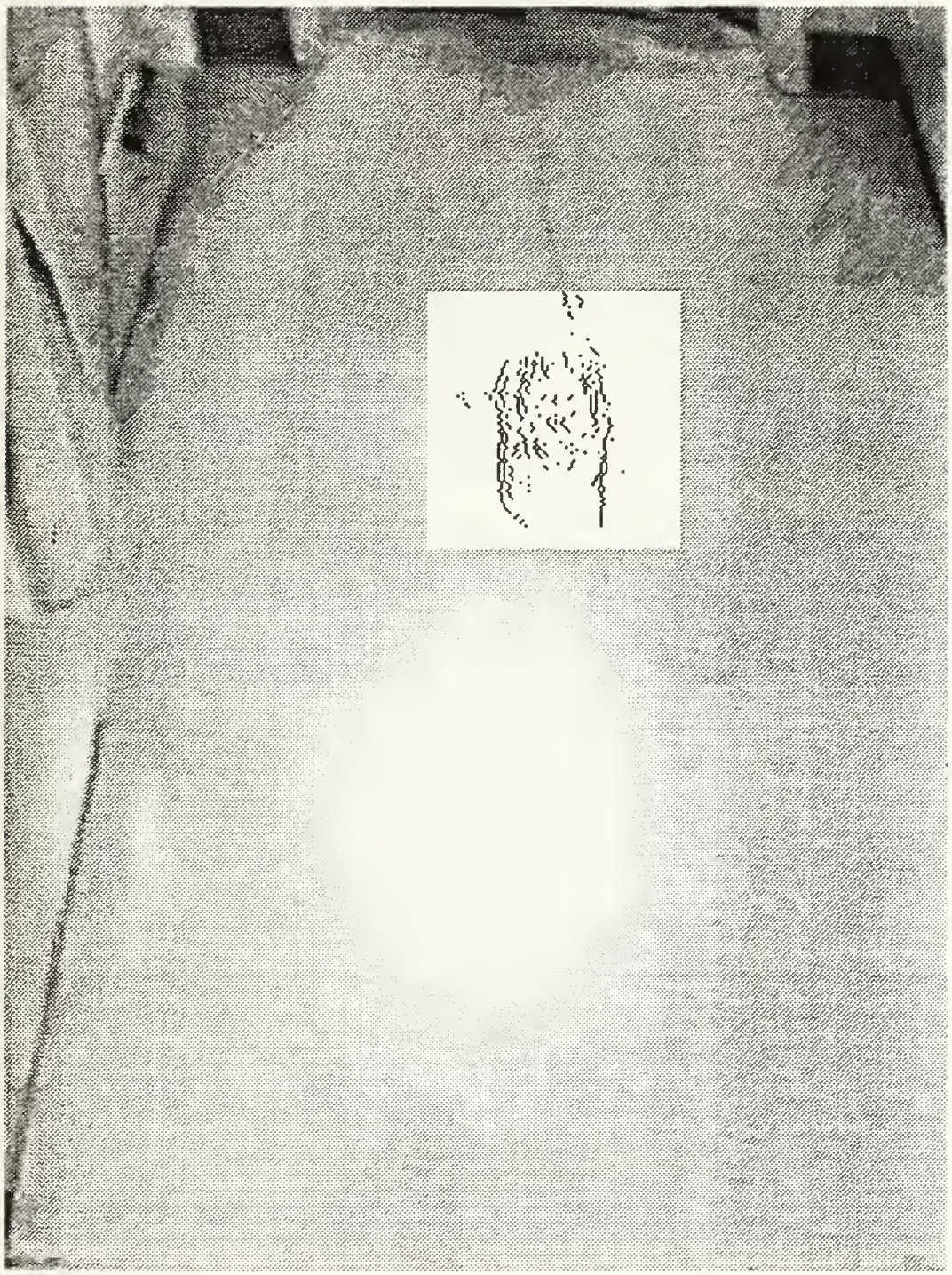

Figure 12. Object After Edge Detection

The source code is written such that each pixel value within the window is examined only once. This represents a marked improvement over other edge detectors, such as the Laplacian. The Laplacian edge detector examines each pixel value nine times and the center of mass algorithm examines each pixel one more time. For this reason, the tracking algorithm using the Laplacian edge detector could not be implemented in real-time on our system.

The goal of the edge detector is to overcome the problem of intensity variation. Its ability to accomplish this task is used as a test of its reliability. Initially, the edge detector is implemented in conjunction with a homogeneous background. Under this condition, intensity variations do not hinder the algorithm's ability to track the object. However, the edge detector failed to be reliable when implemented in a complex background. The Kalman filter detected all edges within the window, but as the tracked object approached another object, the tracking algorithm was unable to distinguish between the two objects. The-center-of mass algorithm then failed to center on the tracked object and eventually track was lost. Figures 13 and 14 show the edge detection using the Kalman filter of an object in a complex background. Although the edge detector overcame the problem of intensity variation, its inability to track objects in a complex background made it very unsuitable for implementation.

Figure 13. Image with Complex Background
Before Edge Detection

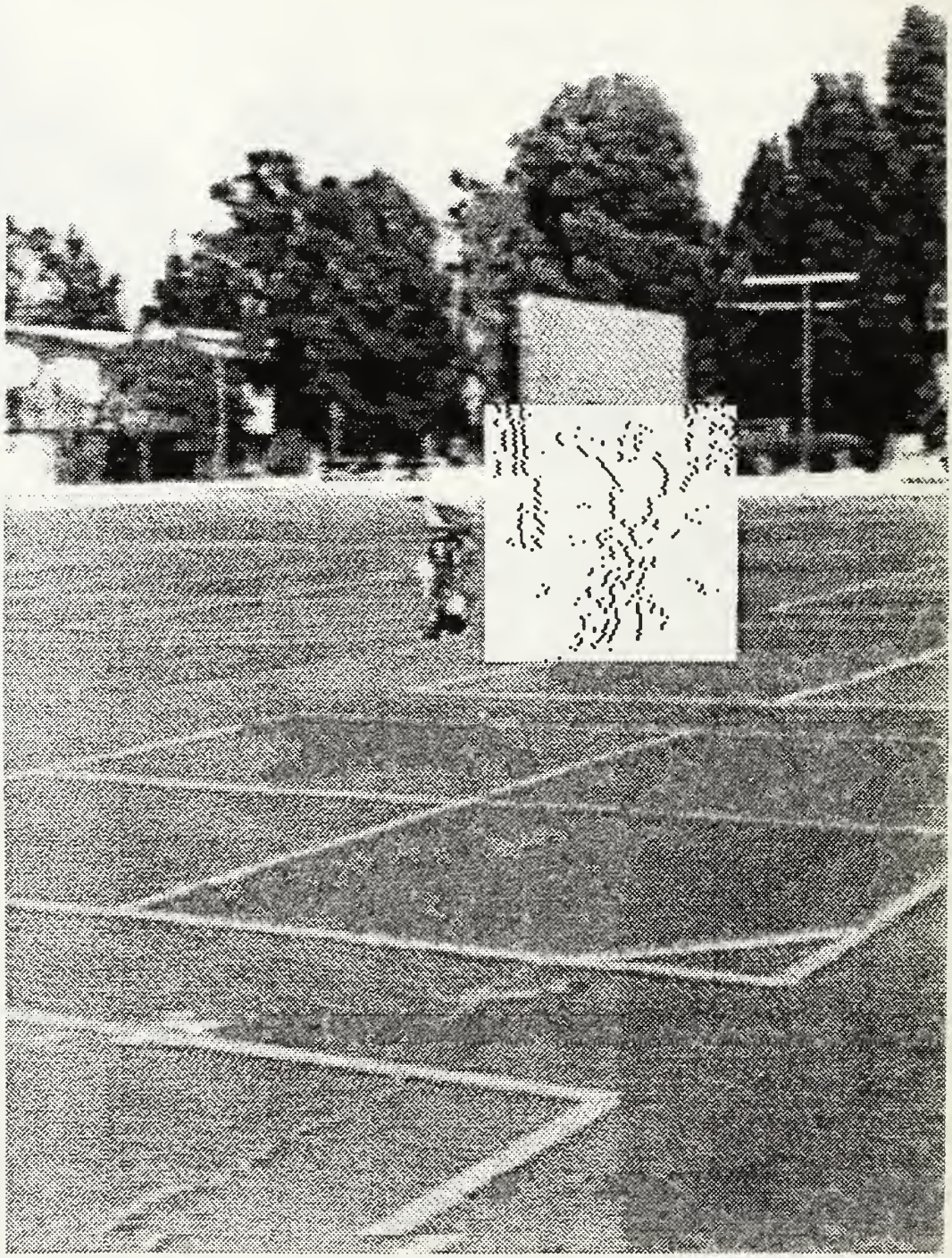

Figure 14. Image with Complex Background
After Edge Detection

V. ADJUSTING THE THRESHOLD RANGE

A. GENERAL

Although the edge detector proved to be effective in the presence of intensity variations, its inability to be used with a complex background made it undesirable to implement in a realistic situation. An alternative to edge detection is to adapt the threshold range discussed in Chapter II to the changing intensity level. This chapter examines two methods for adjusting the threshold range.

B. MOVING AVERAGE

The simplest method for adjusting the threshold range is by a moving-average calculation. During the thresholding operation, the pixel values within the threshold range are averaged to calculate a new mean value for the object's intensity. The threshold range is then centered around the new mean value; this allows the tracker to adjust to slow changes in the intensity level. When implemented, this method adapts well to slow changes in the intensity level of the scene. However, rapid changes in the intensity level due to sudden changes in shading escape the floating thresholding range. Since no pixel value falls within the threshold range when this occurs, the moving average in this case is zero and track of the object is essentially lost.

C. LOST OBJECT CALCULATION

When rapid changes occur, the object's intensity no longer falls within the threshold range. The object appears as background and is lost by the center of mass algorithm. The predictor discussed in Chapter III would indicate that although the object is lost by the algorithm, it should still appear within the window. A moving average of zero indicates that the intensity level has changed too rapidly for the moving average algorithm and that the object has been lost. Upon receiving this indication, a new threshold range is calculated to reflect the intensity change.

Computing the mean intensity value within the window appears to be a viable method of establishing a new threshold range. However, the background intensity values can skew the mean value. If the skew is greater than the tolerance level in the threshold range, the object's intensity level will still be outside of the threshold range and the object will still be undetected. In addition, this method requires that the object be at least one-half the size of the window and that is not always the case.

A variation to this method is to rely on the accuracy of the predictor. Since the predictor proved to be sufficiently accurate in estimating the object location, one can assume that the object is within the range of the window. Therefore, if the mean value is calculated for a much smaller portion at the center of the window (approximately one

fiftieth), the mean value should reflect only the object's new intensity.

This method of adjusting the threshold range is very effective at adapting to rapid intensity variations. During implementation the object being tracked is frequently lost by the center-of-mass algorithm, but the lost-object calculation recomputes the object's intensity and the algorithm continues to track.

A flow chart of the tracking algorithm including the threshold range adjustments is shown in Figure 15. A single image frame is acquired using the snap operation. The image is segmented into a binary image using the threshold operation and each pixel value is subsequently examined to determine to which region (background or object) it belongs. The intensity levels and the x and y coordinates of pixels in the object range are summed for use in the center-of-mass calculation and the moving-average threshold range calculation. The pixels are scanned sequentially across each row and down each column until all pixels within the window have been examined. The center of mass is computed using x and y coordinate sums and the threshold range is adjusted using the moving average technique. When no pixel values fall within the threshold range, a new threshold range is computed using the lost-object calculation. The predictor algorithm and the new center of mass are used to center the window and the process is repeated.

D. UNRESOLVED PROBLEMS

Although the lost-object calculation compensates for rapid intensity variations, it is not sensitive to small differences between the background and the object. When the background intensity is within a few gray level values of the object intensity, the background intensity level falls within the threshold range and the background is perceived as the object. The window centers on a combination of the object and the background, and in most instances, locks onto the background as the object continues to move. This problem occurs primarily when one of two conditions exist. First, under very sunny conditions, background objects that are reflective can be mistaken as the object if the object being tracked is itself bright. The reflection from the background objects are bright spots very similar to the tracked object. Secondly, on a cloudy day, a dark object being tracked can be confused with the background in an excessively shaded area. Under both conditions, the object being tracked remains visible and its motion can be followed by the human operator on the video monitor; but the difference between the intensity levels is so small that the algorithm cannot distinguish between the two.

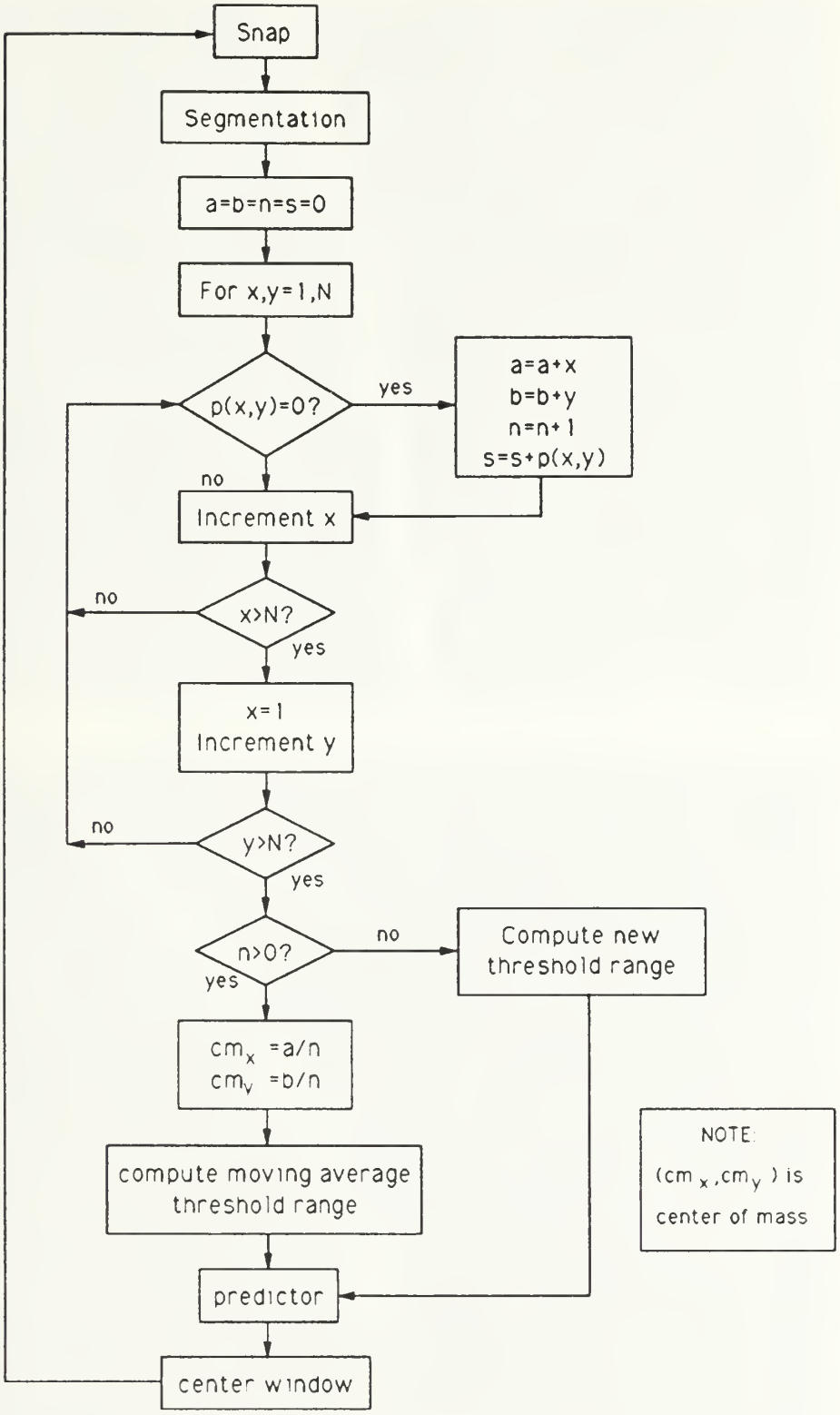

Figure 15. Flowchart of Tracking Algorithm

VI. AUTONOMOUS NAVIGATION

A. GENERAL

Although the Kalman filter edge detector cannot be realistically implemented for tracking moving objects in a complex background, its accuracy and speed as an edge detector warrants further investigation for its use in autonomous navigation. Autonomous navigation represents a situation related, but complementary to, object motion. In autonomous navigation the motion results from the movement of the camera position.

B. THEORY

Assume that a camera is mounted on a moving platform such as a vehicle and is initially directed toward the edge of the road. A window similar to that discussed in Chapter II is superimposed on the input image as shown in Figure 16. The window is free to move horizontally across the image as if the camera is panning, but the vertical movement of the window is fixed to reflect some specified focal point for a given distance in front of the vehicle. As the vehicle moves forward, the image changes to reflect the vehicle's movement and the curvature of the road. When approaching a curve in the road, the image has an appearance similar to Figure 17. The window can track the edge of the road by implementing the

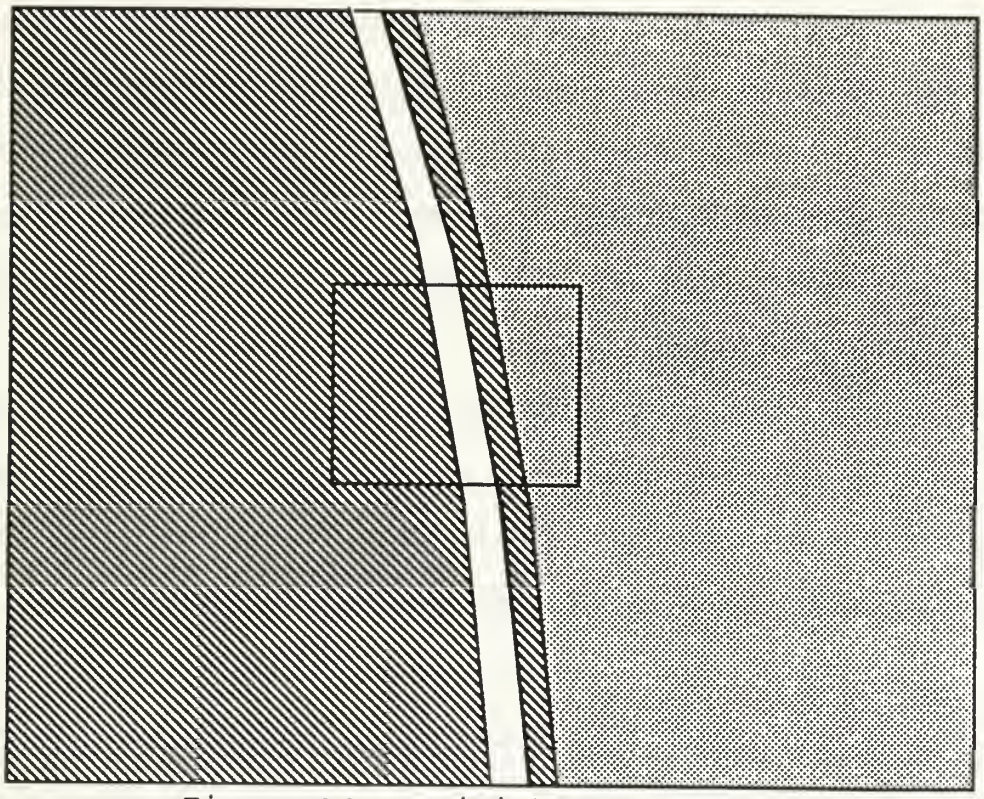

Figure 16. Initial Road Edge

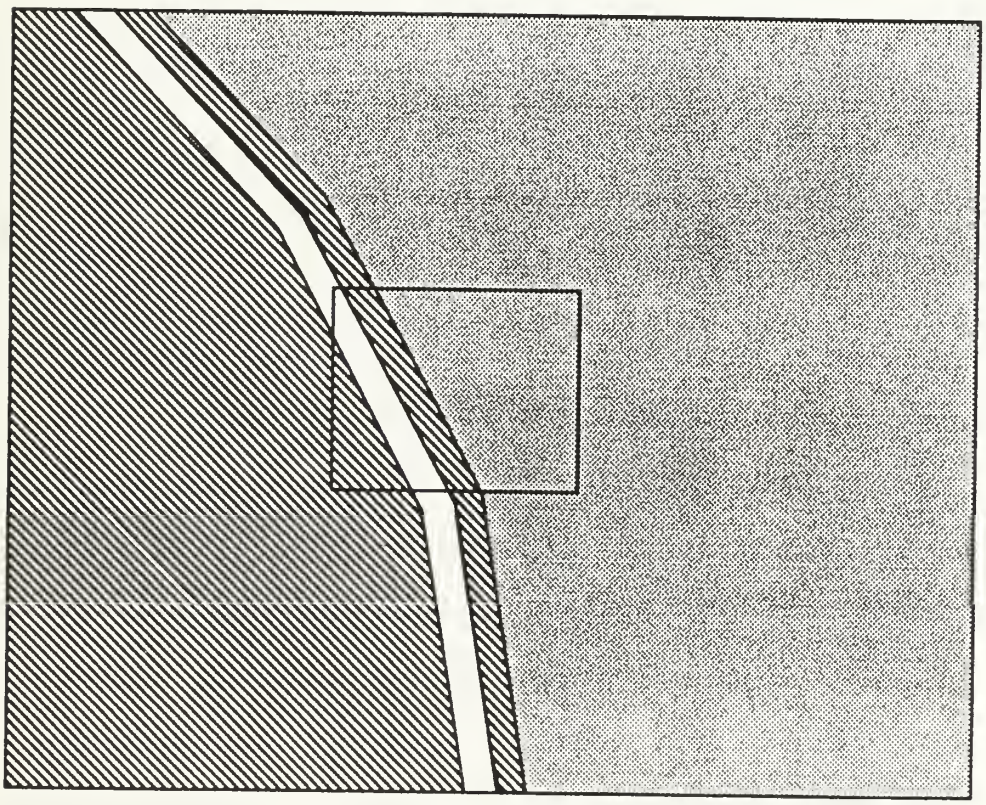

Figure 17. Road Edge at a Curve

Kalman filter edge detector in conjunction with the center-of-mass algorithm discussed in Chapter IV. Centering the box on the edge of the road is equivalent to panning the camera to center the road edge. However, if the camera is mounted rigidly to the vehicle, the vehicle will have to turn in order to maintain the road edge in the center of the window. The output of the algorithm can thus be used as a control input to steer the vehicle. Given a sufficiently accurate edge tracking algorithm, the vehicle should be able to navigate along the road without any human intervention.

C. APPLICATION

The algorithm used to test this autonomous navigation concept is identical to the tracking algorithm described in Chapter IV with one exception. In the tracking application the algorithm is free to move in all directions, while in the autonomous navigation application, vertical movement is restricted to reflect the vehicle's ability to manuever only in the horizontal direction. The input source used to test the algorithm's performance is a video tape taken from a car traveling at normal speeds along various roads.

The algorithm appears to be effective at following the edge of the road and is not adversely affected by a complex background since the camera is kept pointed toward the road edge. At a typical road intersection no edge will be detected, since there are no berm or line markings. The algorithm provides no centering input in this case and the

window remains at the previous location. Therefore, as long as the intersection does not occur on a sharp curve, the algorithm would continue to supply control inputs once the vehicle has passed the intersection and a berm or line marking is again present. The effectiveness of the algorithm can be increased by using additional cameras focused on the opposite edge of the road and/or on the center line. Figure 18 is an example of such an implementation. The additional cameras increase the overall accuracy of the autonomous navigation algorithm since there are three edge detectors, one for each camera. The three edge detectors will complement each other and reduce the possibility of an erroneous control input that may occur when an intersection is on a curve. The three edge detectors may have different variance parameters and tolerence levels to reflect the possibility of differences in the road edges and road surface. The speed and accuracy of the algorithm makes it a very viable method for autonomous navigation. Figures 19 shows a typical road edge after edge detection for a windowed area.

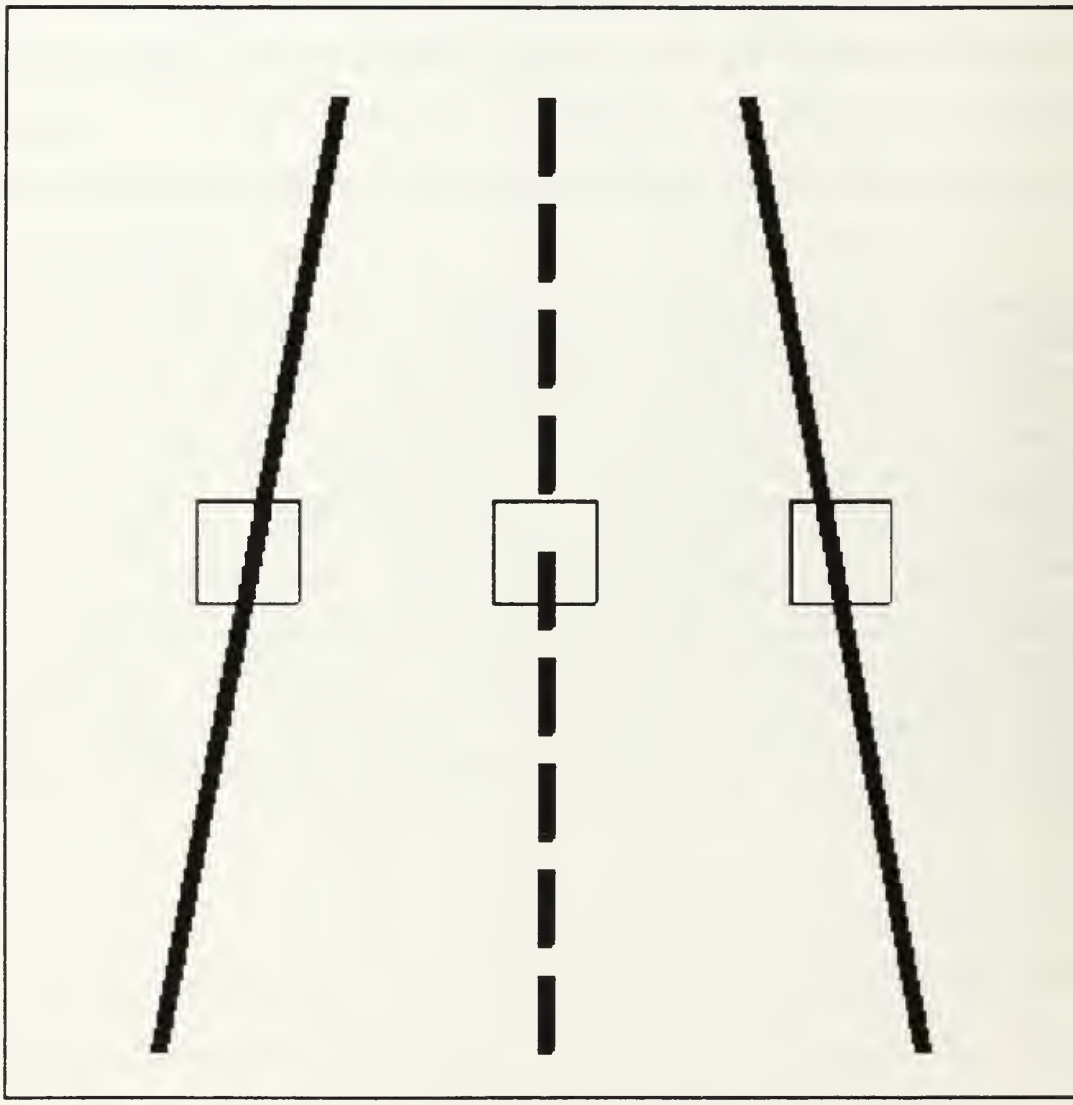

Figure 18. Three Camera Implementation

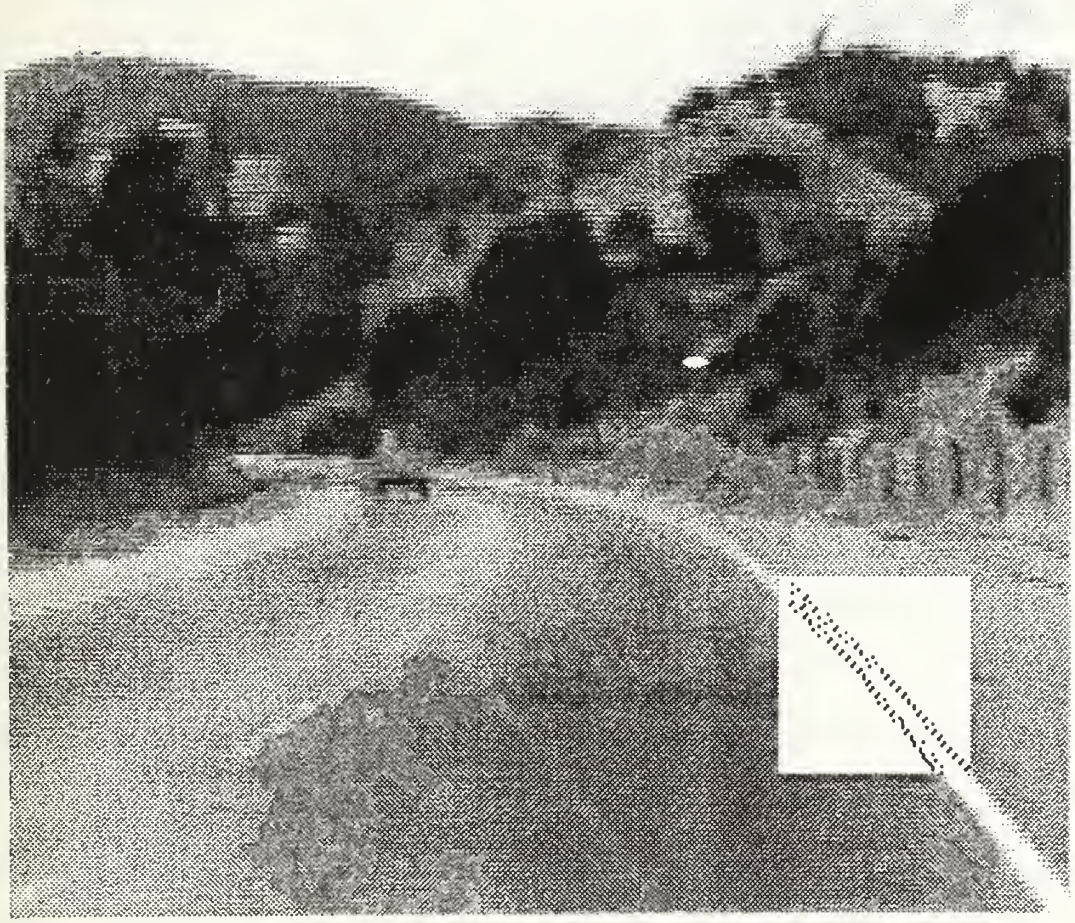

Figure 19. Road Edge After Edge Detection

VII. CONCLUSIONS

The problem of tracking an object in real-time is primarily one of speed and accuracy. The speed at which processing can be completed determines whether real-time image processing can be maintained.

The simplicity of the center-of-mass algorithm allows for a minimum amount of calculations, thus minimal processing time. Its simplicity also permits additional algorithms to be implemented simultaneously in order to enhance its performance.

In this research we have shown that the tracking algorithm with the predictor and moving-average algorithms, together with the lost-object calculation, performs satisfactorily. Tracking is lost only when background and object intensity difference are less than the tolerance within the threshold range. This occurs primarily in bright, sunny conditions for a light-colored object and under dark, cloudy conditions for a dark object. The operator's inability to designate fast moving objects limits the speed of the objects to reasonable video velocities. Once the object is designated, the speed is virtually unlimited.

The introduction of the Kalman filter as an edge detector provides some promising results. The Kalman filter is far superior to other edge detectors in terms of processing speed. Although it is not as accurate as most derivative

operator edge detectors, its ability to detect edges is precise enough for many applications. Initial results of its potential as a means of autonomous navigation is promising. Its capability in this area of image processing and control is one that merits further investigation.

The microcomputer package developed is relatively inexpensive, portable, and easily modified to incorporate any image processing function. The menu system provides an easy means to implement developed algorithms as well as common image processing routines that are contained within the ITEX library. The menu system can be used for detailed analysis of a single image frame or to investigate the application of image processing algorithms in real-time.

APPENDIX A

MICROCOMPUTER PACKAGE

A. GENERAL

The microcomputer package we developed is a collection of subroutines that can be selected from a mouse-driven menu. The menu system used to implement the routines is a software package developed by C Source, Inc. This software package contains graphics, font, and menu libraries which may be utilized to design various displays. This chapter describes the microcomputer package and the routines it incorporates.

B. IMAGE PROCESSING SYSTEM

The components of the image processing system are shown in Figure 20 [Ref. 5]. The image originates at a video source such as a camera or a video tape player. The analog video signal is sent to the framegrabber (enclosed within the dashed line in Figure 20) where it is initially digitized by an analog-to-digital (A/D) converter. The digitized signal is passed through an input lookup table (LUT) which maps intensity levels to values programmed by the operator. The transformed signal is then stored in the frame memory of the framegrabber.

The framegrabber has two available memories which can be selected by the computer. The contents of the frame memory are sent to the video monitor for display through an output lookup table (which operates in the same manner as the input

lookup table) and a digital-to-analog (D/A) converter. The contents of the lookup tables and the frame memories are accessible from the computer for processing applications. [Ref. 6]

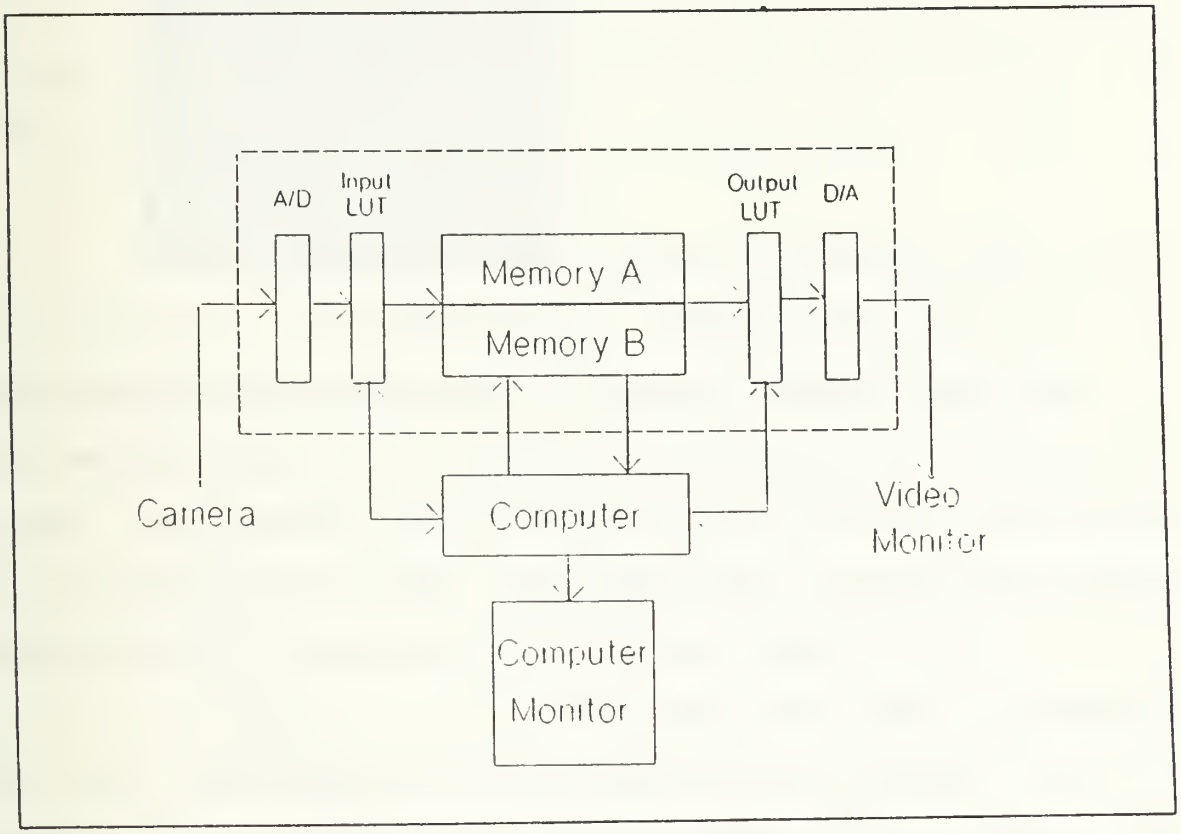

Figure 20. Image Processing System
(from Reference 2)

C. MENU DISPLAY AND SELECTION

The menu begins by displaying the video modes that the system will support. The menu package at this PC system has been hardwired for a VGA video mode to conserve memory. The video mode is easily changed by using the correct video mode number in the source code. Striking any key will cause the

menu display to be activated. Figure 21 is an example of the menu display.

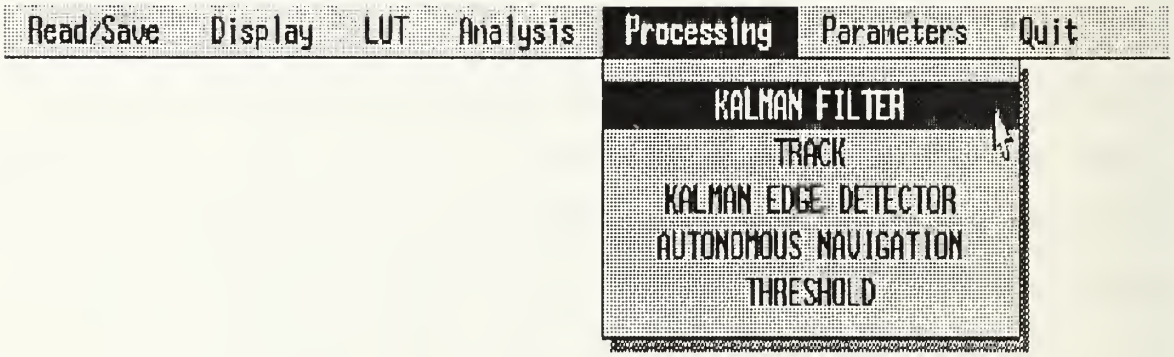

Figure 21. Display Example

The titles across the top of the screen are known as the root menu. The root menu is an easily identifiable title for processing functions that perform similar operations. Below each root menu is a pull-down menu. The pull-down menu is a list of the image processing functions that can be implemented under that root menu.

The selection of a root or pull-down menu may be accomplished by two methods. The cursor may be positioned on the desired menu using the mouse and selection made by pressing the mouse button. Selection may also be accomplished by pressing the hot key (highlighted) of the desired function on the keyboard.

D. MENU SELECTIONS

1. Read/Save

a. Read

Reads an image from a disk file into the Framegrabber memory. Prompts for input of the filename, image size, noise variance of image, and tolerance for filtering.

b. Save

Stores 512 x 512 image displayed on the video monitor into the disk file. Prompts for input of the filename.

2. Display

a. Grab

Continuously acquires image frames from input source and displays them on the video monitor.

b. Snap

Acquires a single 512 x 512 image frame from the input source, stores it in the framegrabber memory, and displays it on the video monitor. Initializes measurement noise covariance, R, to twenty; random forcing function covariance, Q, to one; initial state covariance, P, to four; and the error tolerance level, t, to 1.5.

c. Reinitialize Post-filter

Reinitialize display to original image after using Kalman filter edge detection on a single frame.

d. Reinitialize Post-threshold

Reinitializes display to original image after using thresholding on a single frame.

3. LUT

a. Set LUT

Selects which 256-byte output bank is used for the output video transform. Prompts for input of bank number.

b. Store LUT in Array

Stores LUT values in an array and converts framegrabber memory to reflect LUT transform. Prompts for input of the bank number and queries on area to be converted.

4. Analysis

a. Histogram

Computes and displays the histogram of the image. Queries on the area for which the histogram is to be computed. Strike any key to continue processing.

b. Equalize

Displays histogram equalized image on the video monitor. Queries on the area for which equalization is based and the desired bank for the equalization LUT.

5. Processing

a. Kalman Filter

Filters a designated area of a single frame as discussed in Chapter IV.

b. Track

Tracks a designated object using the center of mass algorithm. The predictor algorithm and intensity variation improvements are incorporated.

c. Kalman Edge Detector

Tracks a designated object using the Kalman filter edge detector discussed in Chapter IV.

d. Autonomous Navigation

Centers a window on a road edge continuously as discussed in Chapter VI.

e. Threshold

Thresholds a single frame of an image. Prompts for input of the threshold value and queries on the area for which thresholding will be done.

6. Parameters

a. Target Area

Designates target area for further processing or analysis. A cursor is displayed on the video monitor for a single image frame. Cursor can be positioned over desired area using the mouse. Depression of the left mouse button will center a rectangle on the target area. The rectangle size can be increased or decreased using the query boxes on the computer screen.

b. Covariance - P

Allows for changes to the covariance, P, of the initial states. Prompts for input of the new value.

c. Covariance - Q

Allows for changes in the covariance, Q, of the random forcing function, F(k). Prompts for input of the new value.

d. Error Tolerance

Allows for changes in the error tolerance level. Prompts for input of the new value. Values are normally between one and two.

e. Covariance - R

Allows for changes to the covariance of the measurement noise, W(k). Prompts for input of the new value.

APPENDIX B

```
                /****************************************
                 *                                      *
                 *              MENU                    *
                 *                                      *
                 ****************************************/

#include "mer_menu.h"                       /* header w/user changes */

     /*  A jump table of the functions called by pull-down menu items.
      *  The position in the array is the same as the associated item's
      *  position in the PDM_DEM array.*/
LOCAL void (*pull_down_vec[])() =
          { read_image, save_image, grab_it, snap_it,
            reinit, thres_reinit, set_lut,store_lut,
            histo_gram, histo_equalize,snap_kalman,
            track_predict,kalman_edge,auto_nav, thres_hold,
            snap_rect,init_p, init_q, tolerance, variance};

     /*  A simple translation table to go from the list of
      *  accepted video modes to the actual bios mode numbers.*/
UTINY xlat_video[] = { 0, 4, 6, 0xD, 0xE, 0x10, 0xF, 0x12, 0x11, 0 };

        /*  Message displayed when the program is started.*/
TEXT *usage_msg[]= {" \n THIS PROGRAM HAS BEEN HARDWIRED",
                    "FOR A VIDEO MODE OF 640x480x16,",
                    "AVAILABLE MODES ARE LISTED BELOW:",
                    " \n  Supported Video Modes",
                    "1. 320x200x4    2. 640x200x2 ",
                    "3. 320x200x16   4. 640x200x16",
                    "5. 640x350x16   6. 640x350x2 ",
                    "7. 640x480x16   8. 640x480x2 ",
                    "          9. Hercules",
                    "",
                    "WARNING: BE SURE YOUR HARDWARE",
                    " CAN HANDLE THE SELECTED MODE.",
                    };

LOCAL INT clear_color, dflt_fh, pd_ascii_code, curr_video_mode;

main(argc, argv)
     INT argc;
     TEXT *argv[];
{
FAST INT i;
EVENT *ev, *get_event();

                /* print out initial message*/
for(i=0;i<(sizeof(usage_msg)/sizeof(TEXT *));puts(usage_msg[i++]));

getch();      /* halts display until a key is hit */

/************* initialization of the frame board ***************/
sethdw(0x100,0xD0000L,DUAL);   /* defines hardware address for
                                  registers and frame memory */
setdim(512,512,8);             /* sets board dimensions */
fgon();                        /* turn on board */
initialize();                  /* initialize all registers */
setvmask(0x00);                /* set the video mask for an aquire */
sclear(255);                   /* clear screen white */
/*****************************************************************/
```

```
init_loop:  /* marker for reinitialization after plotting histogram */

i = 7;      /* Hard wire for 640 X 480 X 16 */

curr_video_mode = xlat_video[i];  /* get true bios mode number */

          /*  Call the local function to set the graphics mode.
           *  This function also determines the appropriate default
           *  font, depending on the video resolution.
**********************************************************************/
SCREEN(0x8000 + curr_video_mode);
dflt_fh = open_mem_font(&iss_st14);
clear_color = 15;
_gfx_color_box(0, 0, _gfx.max_x, _gfx.max_y, clear_color);
/**********************************************************************/

     /* run easy startup routine (in AUTO_MNU.C) for initialization */
init_auto_gfx_menu(1, 0x800, 2, dflt_fh, 640, 480);

           /* take care of variable resolutions and color problems */
reset_auto_cs_layout(YES);

set_root_menu(&root_menu);      /* register & display the root menu */

mouse_show_crsr();              /* show the mouse cursor */

          /***** Continue getting event until exit *****/
/**********************************************************************/
while (1) {
     ev = get_event(1);
     pd_ascii_code = ev->ascii_code;

                    /* determine pull-down menu */
     if (ev->scan_code == PULL_DOWN_SCAN_CODE)
          (*pull_down_vec[ev->ascii_code])();

               /* if quit is select then exit in quit function */
     else if (ev->scan_code == ROOT_MENU_SCAN_CODE) then quit();

          /* if histogram is selected reinitialize after display */
     if(ev->ascii_code== 8)
       {goto init_loop;}
     clear_menus();    /* clears pull-down menus */
     }
}
/**********************************************************************/

                  /***** BYE FUNCTION *****/
/**********************************************************************/
/* This is used to exit the menu system and reset things to normal. */

bye()
   {
    halt_q();         /* decouples from keyboard interrupt */
    fgoff();          /* turn off board */
    SCREEN(0);        /* reset screen mode */
    exit(1);
    }
/**********************************************************************/
```

```
                /******* QUIT MENU ********/
/*************************************************************************/
/* This function displays dialog box to confirm desire to exit menu.*/

TEXT *quit_text[] = {"Are you sure you", "want to leave the program?"};

void quit()
{
GFX_BOX *gbox;
EVENT *ev;

                 /* display dialog box */
gbox = build_auto_box(DBOX0, (TEXT *) 0, 2, dbox_init(quit_text));
ev = set_box_wrt_menu(JUSTIFY_END, 0, 0, gbox);

if (ev->ascii_code == OK_BUTTON) {bye();}   /* exit if button OK */
clear_box();                                /* else clear dialog box */
}
/*************************************************************************/

                /****** READ IMAGE ******/
/*************************************************************************/
/* This function reads in a single frame and initializes parameters */

void read_image()
     {
     int err,view_hand;
     char comment[200];
     INT_PAR *par_ptr;
     INT_SIZE *size_ptr;
     GFX_BOX  *gbox;
     EVENT *ev;
     static char name[20],resp_size[3];
     static char resp_var[6],resp_tol[3];

               /* Form box displayed for input */
     static FORM_ENTRY form[]={ {"Filename" ,name,20},
                                {"Image size",resp_size,3},
                                {"Variance",resp_var,6},
                                {"Tolerance",resp_tol,3}};

     par_ptr=(INT_PAR *)&parameters;   /* structures with parameters*/
     size_ptr=(INT_SIZE *)&image_size; /* and image and window sizes*/

                /***** Display forms box *****/
     gbox=build_auto_box(FBOX0,(TEXT *) 0,2,fbox_init(form));
     ev=set_box_wrt_menu(JUSTIFY_START,0,0,gbox);
     if(ev->ascii_code == CANCEL_BUTTON)
       { clear_box();
        return;
        }
                /*****************************/
                /*** Initialize parameters ***/
     size_ptr->size=atoi(resp_size);             /*   image size    */
     par_ptr->var=atof(resp_var);                /*    R matrix     */
     par_ptr->tol=atof(resp_tol);                /* error tolerance */
     size_ptr->initx=(512-(size_ptr->size))/2;   /*******************/
     size_ptr->inity=size_ptr->initx;            /*                 */
     size_ptr->maxx=size_ptr->size;              /* initial display */
     size_ptr->maxy=size_ptr->maxx;              /* coordinates.    */
```

```
     size_ptr->a=size_ptr->initx;                    /*                  */
     size_ptr->b=size_ptr->inity;                    /*                  */
     size_ptr->dx=(size_ptr->maxx)-(size_ptr->initx);/*                  */
     size_ptr->dy=size_ptr->dx;                      /******************/
     par_ptr->q=1;                                   /*     Q matrix     */
     par_ptr->pinit=4;                               /*     P matrix     */
                  /*****************************/
          /**** read in image using the readim command ****/
     select_mem(MEM_A);                  /*  select frame memory A   */
     if((err==readim(size_ptr->initx,size_ptr->inity,
         size_ptr->dx,size_ptr->dx,name,comment))<0)
       {
        switch(err){
        case FILE_ERROR:
           printf("error opening file\n");
           break;
        case FORMAT_ERROR:
           printf("unknown file format\n");
           break;
        default:
           printf("unknown error\n");
           break;}
        exit(1);
        }
      display_mem(MEM_A);                    /*    display memory A   */
      select_mem(MEM_B);                     /* select frame memory B */
      display_mem(MEM_B);                    /*    display memory B   */
      readim(size_ptr->initx,size_ptr->inity,
         size_ptr->dx,size_ptr->dx,name,comment);
      clear_box();                           /*    clear forms box    */
      }
                  /*****************************/

                    /****** SAVE IMAGE ******/
/*************************************************************************/
/******** This function saves a single frame to a data file. ********/

void save_image()
     {
     int err;
     INT_PAR *par_ptr;
     INT_SIZE *size_ptr;
     GFX_BOX  *gbox;
     EVENT *ev;
     static char name[20],comment[20];
                 /* Forms box displayed for input */
     static FORM_ENTRY form[]={ {"Filename" ,name,20},
                                {"Comment",comment,20}};

     par_ptr=(INT_PAR *)&parameters;   /* structures with parameters*/
     size_ptr=(INT_SIZE *)&image_size; /* and image and window sizes*/

               /* Display forms box */
     gbox=build_auto_box(FBOX0,(TEXT *) 0,2,fbox_init(form));
     ev=set_box_wrt_menu(JUSTIFY_START,0,0,gbox);
     if(ev->ascii_code == CANCEL_BUTTON)
       { clear_box();
        return;
        }
                 /*********************/
```

```
          /* Save image using saveim function */
     if((err==saveim(size_ptr->a,size_ptr->b,
          512,512,EIGHT_BIT,name,comment))<0)
        {
        switch(err)
           {
           case FILE_ERROR:
              printf("error opening file\n");
              break;
           case FORMAT_ERROR:
              printf("unknown format selected\n");
              break;
           default:
              printf("unknown error%d\n");
              break;
           }
        exit(1);
        }
                    /***********************/
     clear_box();     /* clear forms box */
     }
/*************************************************************************/

                    /******* GRAB *******/
/*************************************************************************/
/*** This function continuously acquires image frames...real-time ***/

void grab_it()
     {
     grab(-1);
     }
/*************************************************************************/

                    /******* SNAP ******/
/*************************************************************************/
/* This function acquires a single frame and initializes parameters */

void snap_it()
     {
      INT_PAR *par_ptr;
      INT_SIZE *size_ptr;
      par_ptr=(INT_PAR *)&parameters;  /* structures with parameters*/
      size_ptr=(INT_SIZE *)&image_size;/* and image and window sizes*/

      display_mem(MEM_B);     /* display memory B */
      snap(WAIT);

                    /* Initialize parameters */
      size_ptr->size=512;                          /*   image size    */
      par_ptr->var=20;                             /*    R matrix     */
      par_ptr->tol=1.5;                            /* error tolerance */
      size_ptr->initx=(512-size_ptr->size)/2;      /*******************/
      size_ptr->inity=size_ptr->initx;             /*                 */
      size_ptr->maxx=size_ptr->size-1;             /* Initial display */
      size_ptr->maxy=size_ptr->maxx;               /*                 */
      size_ptr->a=size_ptr->initx;                 /* coordinates     */
      size_ptr->b=size_ptr->inity;                 /*                 */
      size_ptr->dx=size_ptr->maxx-size_ptr->initx;/*                  */
      size_ptr->dy=size_ptr->dx;                   /*******************/
      par_ptr->q=1;                                /*    Q matrix     */
```

```
      par_ptr->pinit=4;                          /*    P matrix     */
      }
/*************************************************************************/

              /******* REINITIALIZE POST-FILTERING ******/
/*************************************************************************/
/* This function will reinitialize the image after filtering to the */
/* original image.                                                   */

void reinit()
     {
     char comment[200];
     INT_PAR *par_ptr;
     INT_SIZE *size_ptr;
     par_ptr=(INT_PAR *)&parameters;   /* structures with parameters*/
     size_ptr=(INT_SIZE *)&image_size; /* and image and window sizes*/

     select_mem(MEM_B);                /* select memory B           */

              /* Read original image from temporary file */
     readim(size_ptr->a,size_ptr->b,size_ptr->maxx,
            size_ptr->maxy,"temp1",comment);

     display_mem(MEM_B);               /* display original image    */
     }
/*************************************************************************/

              /****** REINITAILIZE POST-THRESHOLD ******/
/*************************************************************************/
/* This function will reinitialize the image after thresholding to  */
/* the original image.                                               */

void thres_reinit()
     {
     char comment[200];
     INT_PAR *par_ptr;
     INT_SIZE *size_ptr;
     par_ptr=(INT_PAR *)&parameters;  /* structures with parameters*/
     size_ptr=(INT_SIZE *)&image_size;/* and image and window sizes*/

     select_mem(MEM_B);                    /*    select memory B   */

               /* Read original image from temporary file */
     readim(size_ptr->a,size_ptr->b,size_ptr->maxx,
            size_ptr->maxy,"temp2",comment);

     display_mem(MEM_B);                   /*                      */
     }
/*************************************************************************/

                       /****** SET LUT ******/
/*************************************************************************/
/* This function will let you set the desired output lookup table   */

void set_lut()
     {
     int bank;
     GFX_BOX  *gbox;
     EVENT *ev;
     static char resp_size[1];
```

```
                /* Forms box displayed for input */
     static FORM_ENTRY form[]={ {"Enter bank number" ,resp_size,1}};

                    /* Display forms box */
     gbox=build_auto_box(FBOX0,(TEXT *) 0,2,fbox_init(form));
     ev=set_box_wrt_menu(JUSTIFY_START,0,0,gbox);
     if(ev->ascii_code == CANCEL_BUTTON)
       { clear_box();
         return;
         }

     bank=atoi(resp_size);                          /*     bank number     */
     setlut(GREEN,bank);                            /*       set LUT       */
     clear_box();                                   /*    clear forms box  */
     }
/*********************************************************************/

                  /****** STORE LUT ******/
/*********************************************************************/
/* This function will store a desired LUT trnsformation in the frame*/
/* memory so that transformed values can be filtered, equalized, and*/
/* processed.                                                        */

void store_lut()
     {
     int bank,lut[256];
     int c,x,y,pixarray[512],intens;
     INT_PAR *par_ptr;
     INT_SIZE *size_ptr;
     GFX_BOX  *gbox;
     EVENT *ev;
     static char resp_size[1],resp_tol[1];

                  /* Forms box displayed for input */
     static FORM_ENTRY form[]={{"Default values are:" ,(TEXT *) 0, 0},
                               {"x=0 y=0 dx=511 dy=511",(TEXT *)0, 0},
                               {"Use rectangle values?",resp_tol, 1},
                               {"Enter bank number",resp_size,1},};

     par_ptr=(INT_PAR *)&parameters;   /* structures with parameters*/
     size_ptr=(INT_SIZE *)&image_size; /* and image and window sizes*/

                  /* Display forms box */
     gbox=build_auto_box(FBOX0,(TEXT *) 0,2,fbox_init(form));
     ev=set_box_wrt_menu(JUSTIFY_START,0,0,gbox);
     if(ev->ascii_code == CANCEL_BUTTON)
       { clear_box();
         return;
         }
                  /*********************/

     bank=atoi(resp_size);                    /* bank number  */
     ralut(GREEN,bank,0,256,lut);             /* read LUT     */

                  /* Use default values? */
     if(resp_tol[0] != 'y'){size_ptr->initx=0;size_ptr->inity=0;
                            size_ptr->dx=511;size_ptr->dy=511;}

        /* Transform pixels and resave them in frame memory */
```

```
     for(y=0;y<=size_ptr->dy;y++)
      {
      rhline(size_ptr->initx,size_ptr->inity+y,size_ptr->dx,pixarray);
      for(x=0;x<=size_ptr->dx;x++)
         {
         intens=pixarray[x];
         pixarray[x]=lut[intens];
         }
      whline(size_ptr->initx,size_ptr->inity+y,size_ptr->dx,pixarray);
      }
         /*****************************************************/

     clear_box();                    /* clear forms box */
     }
/*********************************************************************/

                      /****** HISTOGRAM ******/
/*********************************************************************/
/* This function computes the histogram for a selected area and      */
/* display the histogram on the computer monitor. The menu program   */
/* is continued by pressing any key.                                 */

void histo_gram()
     {
     int c,m,view_hand,bad_view;
     long histvals[256];
     float value[256];
     FILE *f_ptr;
     char fname[]="paul";
     int i,mode;
     chartenv env;
     char far *glabel[7]=
      { "0","50","100","150","200","250"," " };
     char far *graylvl[256];
     INT_PAR *par_ptr;
     INT_SIZE *size_ptr;
     GFX_BOX  *gbox;
     EVENT *ev;
     static char resp_var[1];

                    /* Forms box displayed for input */
     static FORM_ENTRY form[]={{"Default values are:" ,(TEXT *) 0, 0},
                           {"x=40 y=45 dx=410 dy=405.",(TEXT *)0,0},
                           {"Use rectangle values?",resp_var, 1},};

     par_ptr=(INT_PAR *)&parameters;   /* structures with parameters*/
     size_ptr=(INT_SIZE *)&image_size; /* and image and window sizes*/

                    /* Display forms box */
     gbox=build_auto_box(FBOX0,(TEXT *) 0,2,fbox_init(form));
     ev=set_box_wrt_menu(JUSTIFY_START,0,0,gbox);
     if(ev->ascii_code == CANCEL_BUTTON)
       { clear_box();          /*************************************/
         clear_menus();        /* If cancel button:clear box and menu,*/
         halt_q();             /* decouple keyboard interrupt, reset  */
         SCREEN(0);            /* screen mode, and return to main.    */
         return;               /*************************************/
        }
                    /*********************/
```

```
                    /* Use default values? */
    if(resp_var[0] != 'y'){size_ptr->initx=45;size_ptr->inity=45;
                           size_ptr->dx=410;size_ptr->dy=405;}

                  /* Compute histogram values */
    histogram(size_ptr->initx,size_ptr->inity,
              size_ptr->dx,size_ptr->dy,1,1,0,histvals);

    clear_box();                              /*     clear forms box    */

    for(m=0;m<=255;m++)                       /* Convert histogram      */
       {                                      /* values to float for    */
       value[m] = ((float)histvals[m]);       /* display.               */
       }

    clear_menus();                            /* Clear menu system for  */
    halt_q();                                 /* display. Decouple key- */
    SCREEN(0);                                /* board and reset screen */
    mode=_VRES16COLOR;                        /* mode. Display uses     */
    while( !_setvideomode(mode))              /* microsoft graphics     */
    mode--;                                   /* fuctions.              */
    if( mode== _TEXTMONO)
    return;
                   /* Microsoft setup for display */
    _pg_initchart();
    _pg_defaultchart(&env,_PG_COLUMNCHART,_PG_PLAINBARS);

    for (i=0;i<256;i++){                 /*********************************/
         graylvl[i] = glabel[6];         /*     Intervals for graph       */
         }                               /*                               */
    graylvl[0] = glabel[0];              /*                               */
    graylvl[50] = glabel[1];             /*                               */
    graylvl[100] = glabel[2];            /*                               */
    graylvl[150] = glabel[3];            /*                               */
    graylvl[200] = glabel[4];            /*                               */
    graylvl[250] = glabel[5];            /*********************************/

    strcpy(env.maintitle.title,"Histogram of Image"); /*************/
    env.maintitle.titlecolor = 6;                     /*   Graph   */
    env.maintitle.justify = _PG_RIGHT;                /*   labels  */
    strcpy(env.subtitle.title," ");                   /*           */
    env.subtitle.titlecolor = 6;                      /*           */
    env.subtitle.justify = _PG_RIGHT;                 /*           */
    strcpy(env.yaxis.axistitle.title,"Intensity");    /*           */
    strcpy(env.xaxis.axistitle.title,"Gray Level");   /*           */
    env.chartwindow.border = FALSE;                   /*************/

    if( _pg_chart( &env,graylvl,value,256))           /*************/
       {                                              /*    Draw   */
       _outtext("Error: can't draw chart");           /*   graph   */
       return;                                        /*************/
       }
    else
       {
       getch();                       /*  Strike any key to continue  */
        }

    }
/*************************************************************************/
```

```
                    /****** EQUALIZE ******/
/*************************************************************************/
/*      This function performs histogram equalization on image.          */
/*      Equalization is based on default values or designated area.      */

void histo_equalize()
     {
     int bank;
     INT_PAR *par_ptr;
     INT_SIZE *size_ptr;
     GFX_BOX  *gbox;
     EVENT *ev;
     static char resp_var[1],resp_tol[1];

                /* Forms box displayed for input */
     static FORM_ENTRY form[]={{"Default values are:" ,(TEXT *)0, 0},
                          {"x=40 y=45 dx=410 dy=405.",(TEXT *) 0, 0},
                          {"Use rectangle values?",resp_var, 1},
                          {"Which bank for LUT?",resp_tol,1}};

     par_ptr=(INT_PAR *)&parameters;   /* structures with parameters*/
     size_ptr=(INT_SIZE *)&image_size; /* and image and window sizes*/

                    /* Display forms box */
     gbox=build_auto_box(FBOX0,(TEXT *) 0,2,fbox_init(form));
     ev=set_box_wrt_menu(JUSTIFY_START,0,0,gbox);
     if(ev->ascii_code == CANCEL_BUTTON)
       { clear_box();
        return;
        }

                     /* Use default values? */
     if(resp_var[0] !='y'){size_ptr->initx=45;size_ptr->inity=45;
               size_ptr->dx=410;size_ptr->dy=405;}

     bank=atoi(resp_tol);      /* Bank number */

                  /* Perform histogram equalization */
     histeq(GREEN,bank,size_ptr->initx,size_ptr->inity,
            size_ptr->dx,size_ptr->dy);

     clear_box();   /* Clear forms box */
      }
/*************************************************************************/

                   /****** KALMAN FILTER ******/
/*************************************************************************/
/* This function performs kalman filter operation on a single frame.*/
/* Filtered image is displayed on video monitor.                    */

TEXT *form3[]={"Image is being",         /* Text displayed in */
               "filtered on the",        /* dialog box during */
               "video monitor"};         /* filtering.        */

void snap_kalman()
     {
     int i,x,y,pixarray[512];
     float p[512],k[512],error,xold,xnew;
     GFX_BOX  *gbox;
     EVENT *ev;
```

```
INT_PAR *par_ptr;
INT_SIZE *size_ptr;

par_ptr=(INT_PAR *)&parameters;   /* structures with parameters*/
size_ptr=(INT_SIZE *)&image_size; /* and image and window sizes*/

i=0;
p[0]=par_ptr->pinit*par_ptr->var;              /* initialize p */

       /* Save image in temporary file for reinitialization */
saveim(size_ptr->a,size_ptr->b,size_ptr->maxx,
      size_ptr->maxy,EIGHT_BIT,"temp1"," ");

       /* calculate kalman gains and riccati equation */
while(i<=size_ptr->size-2)
   {
    k[i]=p[i]/(p[i]+par_ptr->var);       /* kalman filter gain */
    p[++i]=p[i]-k[i]*p[i]+par_ptr->q;    /* ricatti equation   */
    }

mouse_hide_crsr();

              /* Display dialog box */
gbox=build_auto_box( IMMEDIATE_RETURN + DBOX0,(TEXT *) 0,
                     0,dbox_init(form3));
ev=set_box_wrt_menu(JUSTIFY_END,0,0,gbox);

          /* apply kalman filter to image */
for(y=0;y<=size_ptr->dy;y++)              /* row loop */
 {
 i=0;

 rhline(size_ptr->initx,size_ptr->inity+y,size_ptr->dx,pixarray);
 xold=pixarray[0];           /* make intensity floating point */

 for(x=0;x<=size_ptr->dx;x++)           /* column loop */
    {
     xnew=xold+k[i]*(pixarray[x]-xold);  /* kalman  estimate   */
     error=(pixarray[x]-xnew)/sqrt(p[i]+par_ptr->var);/* error */

     i++;                                /*increment kalman gains */
     if(fabs(error)>=par_ptr->tol)       /*   confidence interval */
       {
        xnew=pixarray[x];        /* reinitialize kalman filter */
        i=0;
        pixarray[x]=0;
        }
      else
        {
        pixarray[x]=255;
    /*    pixarray[x]=xnew;*/             /* make estimate integer */
        }

      xold=xnew;                /* estimate is old value for next
                                     iteration */
      }

          /* display filtered image */
 whline(size_ptr->initx,size_ptr->inity+y,size_ptr->dx,pixarray);
```

```
      }
    clear_box();
    mouse_show_crsr();
    }
/*************************************************************************/

                        /****** TRACK ******/
/*************************************************************************/
/* This function will track an object using the center of mass      */
/* technique with a predictor algorithm, a moving average threshold */
/* range, and a lost object calculation included.                    */

void track_predict()
     {
     int box_size,target,x,y,pixarray[55];
     int bl,k,center,m3,m4,coord[2];
     int target_min,target_max;
     float sumx,sumy,sumt,target_sum,center_x,center_y;
     float coord_predict[2],vel[2],count,sum;
     float kgain[20][2];

     kalman_gain(kgain);      /* Compute kalman gains for predictor */

     /********** Loop until both mouse buttons are pressed **********/
     while(!(mouse_state.buttons & 0x01))
          {
          grab(-1);                              /* Real-time image */
          real_mouse(coord);                     /* designate object */
          box_size=47;
          target_sum=0.0;

          for(y=coord[1]-2;y<coord[1]+3;y++)    /*********************/
             {
             for(x=coord[0]-2;x<coord[0]+3;x++)
                {                                /*   compute mean   */
                target_sum+=rpixel(x,y);         /*  intensity value */
                }
             }
          target=(target_sum/25)+.5;             /*********************/

          center=box_size/2.0;
          bl=box_size-2;
          k=0;
          coord_predict[0]=coord[0];           /* initialize object */
          coord_predict[1]=coord[1];           /* location.         */
          vel[0]=0.0;
          vel[1]=0.0;

          rectangle(coord[0]-center,coord[1]-center,
                    box_size,box_size,0);

          target_min=target-4;                /* set threshold range */
          target_max=target+4;

          /******* Loop until right mouse button is pressed ********/
          while(!(mouse_state.buttons & 0x02))
              {
              snap(WAIT);
              sumx=0.0;sumy=0.0;
              sumt=0.0;count=0.0;
```

```
    for(y=1;y<=bl;y+=4)                    /* row loop  */
       {
       rhline(coord[0]-center,coord[1]-center+y,
       box_size,pixarray);

       for(x=1;x<=bl;x+=4)                   /* column loop*/
          {
          if((pixarray[x]>=target_min) &&
              (pixarray[x]<=target_max))   /* Object? */
            {
            sumt+=pixarray[x];          /**********************/
            sumx+=x;                    /* Parameters used to*/
            sumy+=y;                    /* calculate center  */
            count++;                    /* of mass.          */
            }                           /**********************/

          else {sum+=pixarray[x];}
          }
       }

    if(count!=0.0){                     /**********************/
      center_x=(sumx/count);            /* Compute center of  */
      center_y=(sumy/count);            /* mass and moving    */
      target_min=(sumt/count)-3.5;      /* average.           */
      target_max=(sumt/count)+4.5;      /**********************/
      }
      else{
        center_x=center+1;              /**********************/
        center_y=center+1;              /* Compute lost object*/
        target_sum=0.0;                 /* mean value.        */

        for(y=coord[1]-4;y<coord[1]+5;y++)
           {
           for(x=coord[0]-4;x<coord[0]+5;x++)
              {
              target_sum+=rpixel(x,y);
              }
           }

        target=(target_sum/81)+.5;
        target_min=target-4;
        target_max=target+4;
        }                               /**********************/

          /*** Predictor ***/
coord_predict[0]+=.6*vel[0]+kgain[k][0]*(center_x+1-center);
coord_predict[1]+=.6*vel[1]+kgain[k][0]*(center_y+1-center);
vel[0]+=kgain[k][1]*(center_x+1-center);
vel[1]+=kgain[k][1]*(center_y+1-center);

 k++;                       /* increment gains    */
 if(k>19)                   /* reinitialize gains */
    k=0;

 coord[0]=coord_predict[0]+.5;    /* change predicted     */
 coord[1]=coord_predict[1]+.5;    /* value to integer.    */

 rectangle(coord[0]-center,coord[1]-center,
           box_size,box_size,0);  /* Center window        */
```

```
          mouse_get_crsr();                /* button pressed?      */
          }
          /******************* End track loop ***********************/

     mouse_get_crsr();                /* button pressed?      */
     }
     /********************** End function loop ***********************/

     }
/*********************************************************************/

                /****** KALMAN EDGE DETECTOR ******/
/*********************************************************************/
/* This function will track an object using a Kalman filter edge     */
/* detector. A predictor for object location is also included.       */

void kalman_edge()
     {
     int box_size,i,x,y,pixarray[512];
     int bl,k,center,m3,m4,coord[2];
     float sumx,sumy,center_x,center_y;
     float coord_predict[2],vel[2],count,sum;
     float kgain[20][2],kal_gain_pos[20],kal_gain_vel[20];
     float p[65],ks_gain[65];
     float error,xold,xnew,xhat,yhat,velx,vely;

     kalman_gain(kgain);              /* Kalman gains for predictor */
     kalman_scalar(ks_gain,p);        /* Kalman gains for filtering */

     /********** Loop until both mouse buttons are pressed **********/
     while(!(mouse_state.buttons & 0x01))
          {
          grab(-1);
          real_mouse(coord);      /* display cursor for designation */

          rect(coord[0],coord[1],kgain,ks_gain,p); /* track function*/
          }
     /*****************************************************************/
     }
/*********************************************************************/

                  /****** TRACK ALGORITHM ******/
/*********************************************************************/
/* This function uses a Kalman filter to track an object.            */

rect(int a,int b,float kgain[][2],float ks_gain[],float p[])
    {
    int m1,m2,bx,by,target,x,y,pixarray_x[65];
    int bxx,k,xs,ys,m3,m4,i,pixarray_y[65],j;
    float sumx,sumy,xo,yo,tol,var,xold_x,xnew_x,error_x;
    float xhat,yhat,velx,vely,xtemp,ytemp,count,xold_y,xnew_y,error_y;
    INT_PAR *par_ptr;
    INT_SIZE *size_ptr;

    par_ptr=(INT_PAR *)&parameters;   /* structures with parameters*/
    size_ptr=(INT_SIZE *)&image_size; /* and image and window sizes*/

    bx=63;                           /* window size */
    by=63;
```

```
xs=bx/2.0;                          /* window center */
ys=by/2.0;
k=0;                                /* initial conditions */
xhat=a;
yhat=b;
velx=0.0;
vely=0.0;
bxx=bx-2;

/************* Loop until right button is depressed *************/
while(!(mouse_state.buttons & 0x02))
     {
     snap(WAIT);                              /* snap single frame */
     rectangle(a-xs,b-ys,bx,by,0);            /* center window     */
     sumx=0.0;sumy=0.0;
     count=0.0;
     for(y=1;y<=bxx;y=y+6)                    /* row loop */
        {
        i=0;
        j=0;
        rhline(a-xs,b-ys+y,bx,pixarray_x);
        xold_x=pixarray_x[1];

        for(x=1;x<=bxx;x=x+4)                  /* column loop */
           {
                    /* filter image */
           xnew_x=xold_x+ks_gain[i]*(pixarray_x[x]-xold_x);
                   /* compute error */
           error_x=(pixarray_x[x]-xnew_x)/sqrt(p[i]+par_ptr->var);
           i++;

           if(fabs(error_x)>=par_ptr->tol)   /* edge? */
             {
             xnew_x=pixarray_x[x];    /* reinitialize Kalman */
             i=0;
             sumx+=x;
             sumy+=y;
             count++;
             }

           xold_x=xnew_x;
           }                                    /* end column loop */
     }                                          /* end row loop    */

     if(count!=0.0)                             /* edge detected?  */
       {
       xo=(sumx/count);                         /* compute center  */
       yo=(sumy/count);
       }
     else
       {
       xo=xs;
       yo=ys;
       }

     if(k=0)            /* velocity boost for first calculation */
       {
       velx=(xo-xs)*1.5;
       vely=(yo-ys)*1.5;
       }
```

```
        xtemp=xhat+.6*velx+kgain[k][0]*(xo-xs);      /* predictor */
        ytemp=yhat+.6*vely+kgain[k][0]*(yo-ys);
        velx+=kgain[k][1]*(xo-xs);
        vely+=kgain[k][1]*(yo-ys);
        xhat=xtemp;
        yhat=ytemp;
        k++;

        if(k>18)                    /* reset predictor gains */
           k=0;

        a=xhat+.5;                 /* integers */
        b=yhat+.5;

        mouse_get_crsr();
        }
    mouse_get_crsr();
    }
/*************************************************************************/

                /****** AUTONOMOUS NAVIGATION ******/
/*************************************************************************/
/* This function will track a road edge using a Kalman filter            */

void auto_nav()
     {
     int box_size,i,x,y,pixarray[512];
     int bl,k,center,m3,m4,coord[2];
     float sumx,sumy,center_x,center_y;
     float coord_predict[2],vel[2],count,sum;
     float kgain[20][2],kal_gain_pos[20],kal_gain_vel[20];
     float p[65],ks_gain[65];
     float error,xold,xnew,xhat,yhat,velx,vely;

     Kalman_gain(kgain);              /* Kalman gains for predictor */
     kalman_scalar(ks_gain,p);        /* Kalman gians for filtering */

     /************ Loop until both buttons are depressed ************/
     while(!(mouse_state.buttons & 0x01))
          {
          grab(-1);
          real_mouse(coord);          /* cursor for designation */
          rect1(coord[0],coord[1],kgain,ks_gain,p);    /* track */
          }
     /*********************************************************************/
     }
/*************************************************************************/

               /******* TRACK ALGORITHM 2 ******/
/*************************************************************************/
/* This function tracks a road edge horizontally using a Kalman          */
/* filter. A predictor is included for centering.                         */

rect1(int a,int b,float kgain[][2],float ks_gain[],float p[])
     {
     int m1,m2,bx,by,target,x,y,pixarray_x[65];
     int bxx,byy,k,xs,ys,m3,m4,i,pixarray_y[65],j;
     float sumx,sumy,xo,yo,tol,var,xold_x,xnew_x,error;
     float xhat,yhat,velx,vely,xtemp,ytemp,count;
```

```
INT_PAR *par_ptr;
INT_SIZE *size_ptr;

par_ptr=(INT_PAR *)&parameters;   /* structures with parameters*/
size_ptr=(INT_SIZE *)&image_size; /* and image and window sizes*/

bx=43;                            /* window size */
by=45;
xs=bx/2.0;                        /* window center */
ys=by/2.0;
k=0;                              /* initial conditions */
xhat=a;
velx=0.0;
byy=by-2;
bxx=bx-2;

/********* Loop until right mouse button is depressed **********/
while(!(mouse_state.buttons & 0x02))
     {
     snap(WAIT);                        /* snap single frame */
     rectangle(a-xs,b-ys,bx,by,0);      /* center window     */
     sumx=0.0;sumy=0.0;
     count=0.0;

     for(y=1;y<=byy;y=y+6)              /* row loop */
        {
        i=0;
        j=0;
        rhline(a-xs,b-ys+y,bx,pixarray_x);
        xold_x=pixarray_x[1];

        for(x=1;x<=bxx;x=x+4)           /* column loop */
           {
                          /* filter image */
           xnew_x=xold_x+ks_gain[i]*(pixarray_x[x]-xold_x);
                          /* error */
           error=(pixarray_x[x]-xnew_x)/sqrt(p[i]+par_ptr->var);
           i++;

           if(fabs(error)>=par_ptr->tol)  /* edge? */
               {
               xnew_x=pixarray_x[x];    /* reset filter gains */
               i=0;
               sumx+=x;
               count++;
               }
            xold_x=xnew_x;
            }                                /* end column loop */
        }                                    /* end row loop */

        if(count!=0.0)                       /* edge detected? */
          {
          xo=(sumx/count);                   /* compute center */
          }
        else
          {
          xo=xs;
          }

        if(k=0)                        /* initial velocity boost */
```

```
                {
                velx=(xo-xs)*1.5;
                }

             xtemp=xhat+.6*velx+kgain[k][0]*(xo-xs); /* predictor */
             velx+=kgain[k][1]*(xo-xs);
             xhat=xtemp;
             k++;

             if(k>18)                    /* reset predictor gains */
                k=0;

             a=xhat+.5;
             mouse_get_crsr();
             }
          mouse_get_crsr();
          }
/*********************************************************************/

                     /****** THRESHOLD ******/
/*********************************************************************/
/* This function performs a threshold operation on a selected area   */
/* of an image.                                                      */

void thres_hold()
     {
     int thres,c,y,x,pixarray[512],view_hand;
     INT_PAR *par_ptr;
     INT_SIZE *size_ptr;
     GFX_BOX  *gbox;
     EVENT *ev;
     static char resp_var[3],resp_tol[1];

                      /* Forms box displayed for input */
     static FORM_ENTRY form[]={{"Enter threshold value" ,resp_var,3},
                           {"Default values are:" ,(TEXT *) 0, 0},
                           {"x=40 y=45 dx=410 dy=405.",(TEXT *)0,0},
                           {"Use rectangle values?",resp_tol, 1},};

     par_ptr=(INT_PAR *)&parameters; /* structures with parameters*/
     size_ptr=(INT_SIZE *)&image_size;/* and image and window sizes*/

                  /* Display forms box */
     gbox=build_auto_box(FBOX0,(TEXT *) 0,2,fbox_init(form));
     ev=set_box_wrt_menu(JUSTIFY_END,0,0,gbox);
     if(ev->ascii_code == CANCEL_BUTTON)
       { clear_box();
        return;
        }

            /* save image temporary file for reinitialization */
     saveim(size_ptr->a,size_ptr->b,size_ptr->maxx,
            size_ptr->maxy,EIGHT_BIT,"temp2"," ");

                      /* Use default values? */
     if(resp_tol[0] != 'y'){size_ptr->initx=0;size_ptr->inity=0;
               size_ptr->dx=511;size_ptr->dy=511;}

     thres=atoi(resp_var);
     for(y=0;y<=size_ptr->dy;y++)              /* row loop */
```

```
        {
        rhline(size_ptr->initx,size_ptr->inity+y,
               size_ptr->dx,pixarray);

        for(x=0;x<=size_ptr->dx;x++)          /* column loop */
           {
           if(pixarray[x]<=thres){pixarray[x]=50;} /* threshold */
           else{pixarray[x]=200;}
           }
        whline(size_ptr->initx,size_ptr->inity+y,
               size_ptr->dx,pixarray);
        }
     clear_box();               /* clear forms box */
     }
/*********************************************************************/

                  /****** TARGET AREA ******/
/*********************************************************************/
/* This function will allow you to designate a area for processing  */
/* or analysis by placing a cursor on the video monitor.  Pressing  */
/* the left mouse button will cause a window to appear that can be  */
/* increased or decreased in size.                                  */


TEXT *form1[]={"To change rectangle size use" ,   /* forms box info */
               "'I' to increase size",
               "and 'D' to decrease size!",
               "type 's' to stop"};

TEXT *form2[]={"Cursor is on the",              /* forms box info */
               "video monitor!"};

void snap_rect()
     {
     int c,array[2];
     char comment[200];
     GFX_BOX  *gbox;
     EVENT *ev;
     INT_PAR *par_ptr;
     INT_SIZE *size_ptr;

     par_ptr=(INT_PAR *)&parameters;   /* structures with parameters*/
     size_ptr=(INT_SIZE *)&image_size; /* and image and window sizes*/

     display_mem(MEM_B);               /* display original image    */
     select_mem(MEM_B);

          /* save image in temporary file for adjusting window size */
     saveim(size_ptr->a,size_ptr->b,size_ptr->maxx,
            size_ptr->maxy,EIGHT_BIT,"temp"," ");

     select_mem(MEM_A);

     readim(size_ptr->a,size_ptr->b,size_ptr->maxx,size_ptr->maxy,
            "temp",comment);            /* read image into memory A */

     select_mem(MEM_B);

     mouse_hide_crsr();
```

```
                /* Display forms box */
gbox=build_auto_box( IMMEDIATE_RETURN + DBOX0,(TEXT *) 0,
                     0,dbox_init(form2));
ev=set_box_wrt_menu(JUSTIFY_END,0,0,gbox);

nonreal_mouse(array);   /* Display cursor on video monitor */
clear_box();            /* clear forms box */
mouse_show_crsr();

size_ptr->initx=array[0]-32;    /* window location */
size_ptr->inity=array[1]-32;
size_ptr->dx=64;
size_ptr->dy=64;
rectangle(size_ptr->initx,size_ptr->inity,
          size_ptr->dx,size_ptr->dy,0);   /* draw window */


            /* Display forms box */
gbox=build_auto_box( DBOX0,(TEXT *) 0,2,dbox_init(form1));
gbox->buttons = &select_prect_button;
ev=set_box_wrt_menu(JUSTIFY_END,0,0,gbox);
if(ev)
  {

  /************* Loop until OK is selected ***************/
  while(ev->ascii_code)
       {
       erase();             /* erase old window */
       switch(ev->ascii_code)
             {
             case REC_INC:    /* increase window size */
               if(((size_ptr->dx)+(size_ptr->initx)+5>490)
                    || ((size_ptr->initx)-5<20))
                  break;
               else if(((size_ptr->dy)+(size_ptr->inity)+5>490)
                        || ((size_ptr->inity)-5<20))
                  break;  /* limit window to video monitor */
               else
                  ((size_ptr->dx)+=10;
                   (size_ptr->dy)+=10;
                   (size_ptr->inity)-=5;
                   (size_ptr->initx)-=5;)
                   break;
             case REC_DEC:          /* decrease window size */
               (size_ptr->dx)-=10;
               (size_ptr->dy)-=10;
               (size_ptr->initx)+=5;
               (size_ptr->inity)+=5;
               if(size_ptr->dx<10)
                 {size_ptr->dx=10;
                 size_ptr->dy=10;}
               break;
             }

       rectangle(size_ptr->initx,size_ptr->inity,
                 size_ptr->dx,size_ptr->dy,0); /* draw window */

       ev=revisit_box();        /* get new button designation */
       }
```

```
      erase();              /* clear window */
      }
    clear_box();            /* clear forms box */
    }
 /*********************************************************************/

                  /****** COVARIANCE P ******/
/*********************************************************************/
/* This function allows you to change the covariance P.              */

void init_p()
    {
    INT_PAR *par_ptr;
    INT_SIZE *size_ptr;
    GFX_BOX  *gbox;
    EVENT *ev;
    static char resp_var[6];

                  /* Forms box displayed for input */
    static FORM_ENTRY form[]={ {"Enter new P factor:",resp_var,6}};

    par_ptr=(INT_PAR *)&parameters;   /* structures with parameters*/
    size_ptr=(INT_SIZE *)&image_size; /* and image and window sizes*/

                /* Display forms box */
    gbox=build_auto_box(FBOX0,(TEXT *) 0,2,fbox_init(form));
    ev=set_box_wrt_menu(JUSTIFY_END,0,0,gbox);
    if(ev->ascii_code == CANCEL_BUTTON)
      { clear_box();
        return;}

    par_ptr->pinit=atof(resp_var);          /* change value */
    clear_box();                            /* clear forms box */
    }
/*********************************************************************/

                  /****** COVARIANCE Q ******/
/*********************************************************************/
/* This function allows you to change the covariance Q.              */

void init_q()
    {
    INT_PAR *par_ptr;
    INT_SIZE *size_ptr;
    GFX_BOX  *gbox;
    EVENT *ev;
    static char resp_var[6];

                  /* forms box for input */
    static FORM_ENTRY form[]={ {"Enter new Q factor:",resp_var,6}};

    par_ptr=(INT_PAR *)&parameters;   /* structures with parameters*/
    size_ptr=(INT_SIZE *)&image_size; /* and image and window sizes*/

                /* Display forms box */
    gbox=build_auto_box(FBOX0,(TEXT *) 0,2,fbox_init(form));
    ev=set_box_wrt_menu(JUSTIFY_END,0,0,gbox);
    if(ev->ascii_code == CANCEL_BUTTON)
      { clear_box();
        return;}
```

```
     par_ptr->q=atof(resp_var);           /* change value */
     clear_box();                         /* clear forms box */
     }
/*************************************************************************/

                  /****** ERROR TOLERANCE ******/
/*************************************************************************/
/* This function allows you to change the error tolerance.               */

void tolerance()
     {
     INT_PAR *par_ptr;
     INT_SIZE *size_ptr;
     GFX_BOX  *gbox;
     EVENT *ev;
     static char resp_var[6];

                    /* Forms box for input */
     static FORM_ENTRY form[]={ {"Enter new tolerance:",resp_var,6}};

     par_ptr=(INT_PAR *)&parameters;   /* structures with parameters*/
     size_ptr=(INT_SIZE *)&image_size; /* and image and window sizes*/

                    /* Display forms box */
     gbox=build_auto_box(FBOX0,(TEXT *) 0,2,fbox_init(form));
     ev=set_box_wrt_menu(JUSTIFY_END,0,0,gbox);
     if(ev->ascii_code == CANCEL_BUTTON)
       { clear_box();
         return;}

     par_ptr->tol=atof(resp_var);      /* change value */
     clear_box();                      /* clear box    */
     }
/*************************************************************************/

                  /****** COVARIANCE R ******/
/*************************************************************************/
/* This function allows you to change the covariance R.                  */

void variance()
     {
     INT_PAR *par_ptr;
     INT_SIZE *size_ptr;
     GFX_BOX  *gbox;
     EVENT *ev;
     static char resp_var[6];

                     /* Forms box for input */
     static FORM_ENTRY form[]={ {"Enter new R value:",resp_var,6}};

     par_ptr=(INT_PAR *)&parameters;   /* structures with parameters*/
     size_ptr=(INT_SIZE *)&image_size; /* and image and window sizes*/

                    /* Display forms box */
     gbox=build_auto_box(FBOX0,(TEXT *) 0,2,fbox_init(form));
     ev=set_box_wrt_menu(JUSTIFY_END,0,0,gbox);
     if(ev->ascii_code == CANCEL_BUTTON)
       { clear_box();
         return;}
```

```
     par_ptr->var=atoi(resp_var);          /* change value */
     clear_box();                          /* clear forms box */
     }
/*************************************************************************/

                /****** NONREAL_MOUSE ******/
/*************************************************************************/
/* This function displays a cursor on the video monitor for              */
/* designation of a target area.  Press the left mouse button to         */
/* designate target.                                                     */

void nonreal_mouse(int array[])
     {
     int array1[9],array2[9],mx,my;
     mouse_state.x=256;           /* center cursor on video monitor */
     mouse_state.y=256;
     mouse_set_crsr(256,256);

     /********************** Loop until button is depressed *********/
     while(!(mouse_state.buttons))
          {
          mouse_get_crsr();              /* read mouse location */
          mx=mouse_state.x;
          my=mouse_state.y;

          rhline(mx-4,my,9,array1);       /* read pixel values   */
          rvline(mx,my-4,9,array2);       /* for cursor location */

          clr_line(mx-4,my,mx+4,my,0);    /* draw cursor */
          clr_line(mx,my-4,mx,my+4,0);

          whline(mx-4,my,9,array1);        /* write over cursor  */
          wvline(mx,my-4,9,array2);
          }
     array[0]=mx;                         /* return final location */
     array[1]=my;
     }
/*************************************************************************/

                      /****** ERASE ******/
/*************************************************************************/
/* This function reads pixel values from one memory to write over        */
/* the window drawn on the monitor so that the window size can be        */
/* changed.                                                              */

void erase()
     {
     int array1[512],array2[512],array3[512],array4[512];
     INT_PAR *par_ptr;
     INT_SIZE *size_ptr;
     par_ptr=(INT_PAR *)&parameters;  /* structures with parameters*/
     size_ptr=(INT_SIZE *)&image_size;/* and image and window sizes*/

     /* Read pixel values from memory A */
     select_mem(MEM_A);
     rhline(size_ptr->initx,size_ptr->inity,size_ptr->dx,array1);
     rhline(size_ptr->initx,(size_ptr->inity)+(size_ptr->dy),
            size_ptr->dx,array2);
     rvline(size_ptr->initx,size_ptr->inity,size_ptr->dy,array3);
```

```
     rvline((size_ptr->initx)+(size_ptr->dx),size_ptr->inity,
             size_ptr->dy,array4);

     /* Write pixel values in memory B to remove window */
     select_mem(MEM_B);
     whline(size_ptr->initx,size_ptr->inity,size_ptr->dx,array1);
     whline(size_ptr->initx,(size_ptr->inity)+(size_ptr->dy),
            size_ptr->dx,array2);
     wvline(size_ptr->initx,size_ptr->inity,size_ptr->dy,array3);
     wvline((size_ptr->initx)+(size_ptr->dx),size_ptr->inity,
             size_ptr->dy,array4);

     display_mem(MEM_B);
     }
/*************************************************************************/

                        /****** REAL_MOUSE ******/
/*************************************************************************/
/* This function draws a cursor on the video monitor in real time.   */
/* The cursor is used to designate targets.                          */

void real_mouse(int array[])
     {
     int mx,my;
     mouse_state.x=256;                /* center cursor */
     mouse_state.y=256;
     mouse_set_crsr(256,256);

     /******************* Loop until left button is depressed *******/
     while(!(mouse_state.buttons & 0x01))
          {
          mouse_get_crsr();         /* read cursor location */
          mx=mouse_state.x;
          my=mouse_state.y;

          clr_line(mx-4,my,mx+4,my,0);     /* draw cursor */
          clr_line(mx,my-4,mx,my+4,0);
          }
     snap(WAIT);
     array[0]=mx;                          /* return cursor location */
     array[1]=my;
     }
/*************************************************************************/

                        /****** KALMAN_GAIN ******/
/*************************************************************************/
/* This function computes the kalman gains for the predictor.        */

void kalman_gain(float kgain[][2])
   {
   int k;
   float f;
   float pkm[2][2],c[2][2],r,t;
   float ct[2][2],q[2][2],d[2][2],e[2][2];
   float phi[2][2],phit[2][2],pk[2][2],g[2][2];
   INT_PAR *par_ptr;
   INT_SIZE *size_ptr;
   par_ptr=(INT_PAR *)&parameters;     /* structures with parameters*/
   size_ptr=(INT_SIZE *)&image_size;   /* and image and window sizes*/
```

```
    r=5.0;t=.5;
    pkm[0][0]=20.0;pkm[0][1]=20.0;pkm[1][0]=20.0;pkm[1][1]=20.0;
    c[0][0]=1.0;c[0][1]=0;c[1][0]=0;c[1][1]=0;
    phi[0][0]=1.0;phi[0][1]=t;phi[1][0]=0;phi[1][1]=1.0;
    ct[0][0]=1.0;ct[0][1]=0;ct[1][0]=0;ct[1][1]=0;
    q[0][0]=.1;q[0][1]=0;q[1][0]=0;q[1][1]=.1;
    phit[0][0]=1.0;phit[0][1]=0;phit[1][0]=t;phit[1][1]=1.0;

    for(k=0;k<20;k++)
        {
        m_mult(1,2,2,c,pkm,d);        /* Kalman filter equations */
        m_mult(1,2,1,d,ct,e);
        f=1.0/(e[0][0]+r);
        m_mult(2,2,1,pkm,ct,d);
        g[0][0]=f*d[0][0];
        g[1][0]=f*d[1][0];
        kgain[k][0]=f*d[0][0];
        kgain[k][1]=f*d[1][0];
        m_mult(2,1,2,g,c,d);
        d[0][0]=1-d[0][0];
        d[0][1]=-d[0][1];
        d[1][0]=-d[1][0];
        d[1][1]=1-d[1][1];
        m_mult(2,2,2,d,pkm,pk);
        m_mult(2,2,2,phi,pk,e);
        m_mult(2,2,2,e,phit,pkm);
        pkm[0][0]+=q[0][0];
        pkm[1][1]+=q[1][1];
        }
    }
/*************************************************************************/

                        /****** M_MULT ******/
/*************************************************************************/
/* This function multiplies two 2x2 matrices used in the computation*/
/* of the predictor gains.                                               */

void m_mult(int m,int n,int l,float a[][2],float b[][2],float c[][2])
    {
    int y,i,x;

    for(y=0;y<=m-1;y++)
        {
        for(i=0;i<=l-1;i++)
            {
            c[y][i]=0;
            for(x=0;x<=n-1;x++)
                {
                c[y][i]+=a[y][x]*b[x][i];
                }
            }
        }
    }
/*************************************************************************/

                        /****** KALMAN_SCALAR ******/
/*************************************************************************/
/* This function computes the kalman gains for the edge detector.    */

void kalman_scalar(float ksgain[], float p[])
```

```
    {
    int i,q;
    INT_PAR *par_ptr;
    INT_SIZE *size_ptr;
    par_ptr=(INT_PAR *)&parameters; /* structures with parameters*/
    size_ptr=(INT_SIZE *)&image_size;/* and image and window sizes*/

    p[0]=(par_ptr->pinit)*(par_ptr->var);  /* initial P[0] */
    i=0;
    while(i<64)
       {
       ksgain[i]=p[i]/(p[i]+(par_ptr->var));     /* kalman equations */
       p[++i]=p[i]-ksgain[i]*p[i]+(par_ptr->q);
       }
    }
/*************************************************************************/
```

```
/*****************************************
 *                                       *
 *              DATA  FILE               *
 *                                       *
 *****************************************/

#include "mer_menu.h"                  /* Header file with all header */
                                       /* files included plus special */
                                       /* button definitions and list */
                                       /* of all external functions.  */

#define SC          SHOW_CHECK_MARK
#define DC          SC | CHECK_MARK_IS_ON
#define DA          DISABLE_CHOICE
#define SV          SUB_MENU_VECTOR
#define AB          ADD_BAR

                   /* Define size of each pulldown menu */
#define SUB_START        0
#define READ_START       0
#define N_READ           2
#define DSP_START        READ_START + N_READ
#define N_DSP            4
#define LUT_START        DSP_START + N_DSP
#define N_LUT            2
#define ANLY_START       LUT_START + N_LUT
#define N_ANLY           2
#define PRO_START        ANLY_START + N_ANLY
#define N_PRO            5
#define PAR_START        PRO_START + N_PRO
#define N_PAR            5
#define QUIT_START       PAR_START + N_PAR
#define N_QUIT           0

#define TEST_START       8
#define N_TEST           3

BOX_DIM root_bdim;

                  /* Define special button structures */
BUTTON_DATA rect_increase={'I',REC_INC,"INCREASE"};
BUTTON_DATA rect_decrease={'D',REC_DEC,"DECREASE"};
BUTTON_DATA rect_stop={'S',CANCEL_BUTTON,"STOP"};
BUTTON_DATA *rect_btns[]={&rect_increase,&rect_decrease,&rect_stop};
BUTTON_SET select_prect_button={3,rect_btns,&std_btn_layout,
                                &button_cs,-1,2,0,0};

                   /***************************
                    *                         *
                    *    MENU STRUCTURES      *
                    *                         *
                    ***************************/

PDM_DEF choices[] =
    {
     {0,   'R', "READ IMAGE"},                    /* READ/SAVE */
     {0,   'S', "SAVE IMAGE"},

     {0,   'G', "GRAB"},                          /* DISPLAY CHOICES */
     {0,   'P', "SNAP"},
```

```
    {0,   'F', "REINITIALIZE POST-FILTER"},
    {0,   'T', "REINITIALIZE POST-THRESHOLD"},

    {0,   'S', "SET LUT"},                          /* LUT CHOICES */
    {0,   'A', "STORE LUT IN ARRAY"},

    {0,   'H', "HISTOGRAM"},                       /* ANALYSIS CHOICES */
    {0,   'E', "EQUALIZE"},

    {0,   'K', "KALMAN FILTER"},                 /* PROCESSING CHOICES */
    {0,   'R', "TRACK"},
    {0,   'E', "KALMAN EDGE DETECTOR"},
    {0,   'A', "AUTONOMOUS NAVIGATION"},
    {0,   'T', "THRESHOLD"},

    {0,   'T', "TARGET AREA"},                    /* PARAMETER CHOICES */
    {0,   'C', "COVARIANCE - P"},
    {0,   'Q', "COVARIANCE - Q"},
    {0,   'L', "ERROR TOLERANCE"},
    {0,   'V', "COVARIANCE - R"}
  };

                    /****** Root menu ******/
ROOT_DEF pdm[] = { {0, 'R', READ_START,N_READ,"Read/Save",0, 0, 0, 0},
                   {0, 'D', DSP_START, N_DSP, "Display", 0, 0, 0, 0},
                   {0, 'L', LUT_START, N_LUT, "LUT",   0, 0, 0, 0},
                   {0, 'A', ANLY_START, N_ANLY,"Analysis",0, 0, 0, 0},
                   {0, 'P', PRO_START,N_PRO,"Processing", 0, 0, 0, 0},
                   {0, 'M', PAR_START, N_PAR,"Parameters",0, 0, 0, 0},
                   {0, 'Q', QUIT_START, N_QUIT, "Quit",  0, 0, 0, 0}};

MENU_DEF menu = { n_arr_items(pdm, ROOT_DEF), (TEXT *)pdm,
                  &root_layout, &basic_cs, -1, JUSTIFY_START, 3, 0};

MENU_DEF selections = { n_arr_items(choices, PDM_DEF),(TEXT *)choices,
                    &pd_layout, &basic_cs, -1, JUSTIFY_CENTER, 3, 251};

ROOT_MENU root_menu = {0,&menu, &selections, (TEXT *) 0, &root_bdim };

INT_PAR parameters = { 20.0, 1.5, 1.0, 4.0};
INT_SIZE image_size = {512,0,0,512,512,0,0,512,512};
```

```
                /***********************************
                 *                                 *
                 *          HEADER  FILE           *
                 *                                 *
                 ***********************************/

#include <stdio.h>              /* Microsoft header files           */
#include <math.h>
#include <stdlib.h>
#include <conio.h>
#include <ctype.h>
#include <graph.h>
#include <process.h>
#include <time.h>
#include <pgchart.h>
#include <string.h>
#include <bios.h>
#include "gfxf_src.h"           /* Header file w/ GFX defs          */
#include "menu.h"               /* Header file w/ all the MENU defs */
#include "auto_mnu.h"           /* Header file w/ externs for boxes */
#include "itexpfg.h"            /* Header file w/ frame board defs  */
#include <stdtyp.h>

#define U_UP    0x0148             /* Special definitions for buttons */
#define U_DN    0x0150
#define REC_INC 1
#define REC_DEC 2

             /* Structures with parameters for image */
typedef struct _par {float var,tol,q,pinit;} INT_PAR;
typedef struct _im {int size,initx,inity,maxx;
                    int maxy,a,b,dx,dy;} INT_SIZE;

             /* Define function type for box displays */
EVENT *set_box(), *set_box_wrt_menu(), *revisit_box();
GFX_BOX *build_auto_box();

          /* list of every external function used in menu */
extern void sethdw(),kalman_scalar(float ksgain[], float p[]);
extern void setdim(),erase(),kalman_edge(),auto_nav();
extern void histo_gram(), snap_it(), histo_equalize();
extern void snap_rect(), snap_kalman(), thres_hold();
extern void read_image(), save_image(), grab_it();
extern void compress(), variance(), set_lut(), quit();
extern void store_lut(), reinit(), thres_reinit();
extern void init_p(), init_q(), tolerance(), track();
extern void track_predict(),nonreal_mouse(int array[2]);
extern void real_mouse(int array[2]),kalman_gain(float kgain[20][2]);
extern void m_mult(int m,int n,int l,float a[2][2],float b[2][2];
extern void float c[2][2]);
extern GFX_BOX *build_auto_box();
extern EVENT   *set_box_wrt_menu();
extern ROOT_MENU root_menu;
extern BUTTON_SET select_prect_button, select_mem_button;
extern INT_PAR parameters;
extern INT_SIZE image_size;
extern FONT iss_st14;
extern FONT rmn2_9;
extern MENU_DEF selections;
extern MOUSE_STATE _NEAR mouse_state;
```

REFERENCES

1. Gonzalez, Rafael C. and Wintz, Paul, Digital Image Processing, 2d ed., Addison-Wesley Publishing Company, 1987, pp. 331-340.

2. Schalkoff, Robert J. and McVey, Eugene S., "A Model and Tracking Algorithm for a Class of Video Targets", IEEE Transactions on Pattern Analysis and Machine Intelligence, January 1982, volume PAMI-4, number 1, pp. 2-10.

3. Kirk, Donald E., Optimal Estimation: An Introduction to Theory and Applications, Copyright 1975, course notes, pp. 4-32 - 4-41.

4. Anderson, Brian D. and Moore, John B., Optimal Filtering, Prentice-Hall, Inc., 1979, pp. 36-53.

5. Partridge, William J., Real Time Image Enhancement During Underwater Recovery Operations, M.S. Thesis, Naval Postgraduate School, March, 1988.

6. Imaging Technology Incorporated, ITEX PCplus FRAMEGRABBER User's Manual, 1987.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5000	2
3.	Chairman, Code 62 Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
4.	Curricular Officer, Code 32 Naval Postgraduate School Monterey, California 93943-5000	1
5.	Prof. John P. Powers. Code 62Po Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
6.	Prof. Roberto Cristi, Code 62Cx Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	3
7.	Prof. Charles W. Therrien, Code 62Ti Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
8.	Prof. Robert McGhee, 52Mz Computer Science Department Naval Postgraduate School Monterey, California 93943-5000	1
9.	Prof. Anthony J. Healey, Code 69Hy Department of Mechanical Engineering Naval Postgraduate School Monterey, California 93943-5000	1

10. Dr. Rabi Madan 1
Office of Naval Research
800 North Quincy Street
Arlington, Virginia 22247-5000

Thesis
M5402
c.1

Merritt
Video tracking of objects on an enhanced PC system.

Thesis
M5402
c.1

Merritt
Video tracking of objects on an enhanced PC system.

AMERICAN BINDERY
TOPEKA KANSAS

thesM5402
Video tracking of objects on an enhanced
3 2768 000 89558 5
DUDLEY KNOX LIBRARY